General Purpose Deep Learning Accelerator Based on Bit Interleaving

Liang Chang, *Member, IEEE,* Hang Lu, *Member, IEEE,* Chenglong Li, Xin Zhao, *Student Member, IEEE,* Zhicheng Hu, Jun Zhou, *Senior Member, IEEE* and Xiaowei Li, *Senior Member, IEEE*

***Abstract*—Along with the rapid evolution of deep neural networks, the ever-increasing complexity imposes formidable computation intensity on the hardware accelerator. In this paper, we propose a novel computing philosophy called "bit interleaving" and the associate accelerator couple called "*Bitlet*" and *Bitlet-X* to maximally exploit the bit-level sparsity. Apart from the existing bit-serial/parallel accelerators, *Bitlet* leverages the abundant "sparsity parallelism" in the parameters to enforce the inference acceleration. *Bitlet* is versatile by supporting diverse precisions on a single platform, including floating-point 32 and fixed-point from $1b$ to $24b$. The versatility enables *Bitlet* feasible for both efficient inference and training. Besides, by updating the key compute engine in the accelerator, *Bitlet-X* could furthermore improve the peak power consumption and efficiency for the inference-only scenario, with competitive accuracy. Empirical studies on 12 domain-specific deep learning applications highlight the following results: (1) up to $81\times$/$21\times$ energy efficiency improvement for training/inference over recent high-performance GPUs; (2) up to $15\times$/$8\times$ higher speedup/efficiency over state-of-the-art fixed-point accelerators; (3) $1.5mm^2$ area and scalable power consumption from $570mW$ ($fp32$) to $432mW$ ($16b$) and $365mW$ ($8b$) @$28nm$ TSMC; (4) $1.3\times$ improvement of the peak power efficiency for the *Bitlet-X* over *Bitlet*; (5) highly configurable justified by the ablation and sensitivity studies.**

***Index Terms*—Bit-Level Sparsity, Deep Neural Network, Accelerator.**

I. Introduction

With the development of deep neural networks (DNNs) for higher accuracy, the model size is increasing correspondingly, which requires more computational resources for hardware deployment. However, for power-sensitive edge devices, such as drones, power and performance are the limitations for applying bigger-size models to pursue higher accuracy. Accordingly, promoting accelerator efficiency is essential for both high performance and power efficiency. In this work, we focus on leveraging the abundant *bit-level sparsity parallelism* to improve the efficiency of edge- and cloud-based general-purpose deep learning accelerator.

This work was supported in part by the National Natural Science Foundation of China under Grant 62104025 and 62172387, in part by the Youth Innovation Promotion Association of Chinese Academy of Sciences (CAS) under Grant 2021098, in part by the State Key Laboratory of Processors (ICT, CAS) under Grant CARCHB202117 and CLQ202305.

Liang Chang, Chenglong Li, Xin Zhao, Zhicheng Hu, and Jun Zhou are with the University of Electronic Science and Technology of China, Chengdu, China.

Hang Lu and Xiaowei Li are with the State Key Laborotary of Processors, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China; and the Zhongguancun Laboratory, Beijing, China; and the Shanghai Innovation Center for Processor Technologies, Shanghai, China.

There are a lot of works devoted to mining the data sparsity to realize effectual computation [25], [36]. However, sparsity is diverse in different models or even individual network layers in identical models. In addition, sparsity is not always abundant. For example, the zero values in activations are *only* probably to be generated when they are passed through activation functions, such as ReLU. In order to handle this problem, some works aim to recognize the nearly "zero values" in the operands [23], [24] or implement time-consuming sparse (re)training to increase sparsity for pruning [22], [25], [36].

Recently, the headroom of exploiting value-based sparsity has been noticed to the end. From the software perspective, if lossless precision is the first-order design requirement, then there is a visible margin that the compression ratio cannot cross. And the majority of the time is needed to explore such margin to realize the trade-off between the model size and the accuracy. From the hardware implementation perspective, exploiting the value sparsity may lead to a more complicated accelerator design. For instance, enlarging on-chip memory is needed to adapt to the increasing size of the indices [13], [26], [39], [40], which will increase area overhead and power consumption.

In contrast, the "bit-level sparsity" is the inherently more fine-grained sparsity that targets the "zero bits" in each operand. Compared with coarse-grained zero values, higher sparsity can be achieved. Moreover, skipping zero bits in the operand will have no effect on the outcome, which suggests that hardware could be able to directly obtain the bit-level sparsity-based acceleration. In this way, plenty of bit-serial accelerators [7], [17], [21], [30] have been presented. Figure 1 shows a high-level example of the computing paradigms of three different types of accelerator PE. Bit-serial prototypes compute the inner product using numerically identical bit-level arithmetic, with the exception of the early-stage bit-parallel accelerators [10], [16] (Figure 1a). For instance, by organizing and inputting the weights for MAC both in serial fashion, one $8b \times 8b$ product can be divided into eight $1b \times 8b$ products with the identical outcome (Step ❶ in Figure 1b). However, this example shows a serious problem. To mine the maximum potential of the bit sparsity, it is preferable to skip the zero bits as much as possible. Whereas, the locality of the zero bits in each $8b$ operand is unforeseeable, especially after fixed-point quantization. The reason is that quantization will fully utilize the limited bit width to represent the value range, making zero bits arbitrarily interleaved with the essential bit 1s. In order to make full use of bit sparsity, synchronization is essential. It must be carefully implemented as Step ❷ depicts in Figure

1b, before completing the bit-serial MACs in Step ❸.

Prior synchronization schemes include middle-ware-level dense scheduling (*i.e.*, Bit-tactical [21]) and hardware-level direct Booth Encoding (*i.e.* Laconic [29] and Pragmatic [7]) and so on. However, a consistent pattern to define the locality of the sparsity for synchronization using these approaches is difficult to come by. As a potential outcome, the current MAC operation may be suspended in order to align the bit significance, leading to performance degradation. For example, ${w_0}^2$ must wait until ${w_1}^0$ and ${w_1}^1$ have accomplished MAC as shown in Figure 1b. As for the hardware implementation, the design complexity is also increased due to additional circuits for the encoding needed in the Booth encoding. In addition, such serialized organization patterns can not support the floating-point computation and can not be adopted for general-purpose situations.

In this work, we propose a new scheme to investigate the bit-level sparsity dubbed "bit interleaving", which leverages the *sparsity parallelism* presented by a number of weights to accelerate DNNs. Figure 1c illustrates how it organizes the same number of weights in parallel (Step ❶), but differs from the bit serial/parallel accelerator design by interleaving the weights and implementing bit-level MAC in serial (Step ❷). On the one hand, the actual bits used to compute the product are interleaved weights rather than original weights compared with bit parallel accelerators. On the other hand, the serialization technique is expanded to a sequence of interleaved weights together with each independent bit significance, with the exception of the bit-serial accelerators. Considering these two characteristics, bit interleaving is especially attractive to DNNs in three perspectives: ① Bit interleaving is favored by the following behavior in MAC computations: the accumulation targets each independent bit significance, and no synchronization mechanism is necessary as in bit-serial accelerators. ② It can support fixed-point or floating-point computation by configuration and is orthogonal to any quantization/pruning methodology. ③ The "bit interleaving"-directed accelerator design could be used for both inference and training. The following is a list of what we contributed to this work:

- **We propose a novel DNN acceleration pattern called *bit interleaving* that effectively takes advantage of the bit-level sparsity.** Section II reveals the bit sparsity is not only high but also uniform at each bit significance (about 50%). Our observations confirm that such distribution exists: ① in both fixed-point & floating-point weights and ② across big & little models. This *sparsity parallelism* motivates the feasibility and the necessity of the proposed bit interleaving.
- **We propose the corresponding hardware accelerators that maximally mine the potential of bit interleaving.** The accelerators have two members - *Bitlet* and *Bitlet-X*:
  - *Bitlet* serves as the general-purpose deep learning accelerator, which supports both floating-point (fp $32/16$) and fixed-point ($1b$ to $24b$). The sparsity parallelism could all be sufficiently exploited at each bit of significance, regardless of the precision used in practice. Such versatility makes *Bitlet* could bring satisfactory efficiency for both inference and training. As will be shown in Section V, the maximum speedup of *Bitlet* is $15\times$ over existing bit parallel/serial accelerators and $81\times$ over GPUs.
  - *Bitlet-X* serves as the specific accelerator designed for fast and accurate cloud-side inference. By means of clock gating and reducing the wire complexity, we obtain an even faster inference speed and more optimized energy efficiency, but with lossless accuracy on our versatile benchmark suit. Concrete specs: 438 mW power consumption for the floating-point 32[1] inference; 467.58 GOPs/W peak power efficiency, which is $1.3\times$ improvement over the *Bitlet* under 28 nm technology.

TABLE I
ACCELERATOR DESIGN CATEGORIZATION.

Type	Design	Sparsity Exploited	Preci. V.	Training Support
bit parallel	Eyeriss [8], DaDianNao [10]	N/A	16b	No
	Cambricon -S [40], EIE [13]	A-/W-value	16b	No
	SCNN [26]	A-&W-value	16b	No
bit serial	UNPU [22], Stripes [17]	N/A	$1\sim16$b	No
	Bit Fusion [30]	N/A	2,4,8,16b	No
	Pragmatic [7]	A-/W-bit	$1\sim16$b	No
	Bit Tactical [21]	A-bit &W-value	$1\sim16$b	No
	Laconic [29]	A-&W-bit	$1\sim16$b	No
bit inter -leaving	**Bitlet (this work)**	W-bit &W-value, (or A-bit &A-value)	$fp32/16$, $1\sim24$b	Yes

II. MOTIVATION AND RELATED WORK

A. Rethinking Sparsity-aware Accelerators

We categorize the state-of-the-art sparsity-aware accelerators in Table I. In early-stage bit-parallel accelerators, only focus on value sparsity, such as Cambricon series [39], [40] and SCNN [26]. By collaborating with the software-based pruning techniques, more headroom of zero values is created to release the potential of these accelerators. Recently, plenty of bit-serial accelerators pay attention to the bit-level sparsity in genetic activations or weights. The most recent Laconic uses "terms" to extract the essential bits serially after Booth Coding and designs a low-cost LPE to minimize the power increment caused by frequent encoding/decoding [29]. Tactical [21] and Pragmatic [7] use zero-bit skipping to optimize the ineffectual product. Stripes [17] and UNPU [22] implement bit serialization to support fixed-point operands computing without sparsity avoidance. Bit-fusion [30] supports faster spatial and temporal composition to accelerate bit serialization but still does not exploit bit sparsity as well.

The strength of bit serial accelerators is the effectiveness of exploiting the sparsity in bits. However, the bit-serial accelerators provide comparatively lower throughput than their bit-parallel counterpart. Therefore, *bit interleaving* seeks to combine their pros and avoid their cons. Figure 2 shows that the potential of *bit interleaving* speedup could achieve $8\times$ to $29\times$ by exploiting the weight bit and value sparsity (W-bit and

[1]In this manuscript, we interchangeably use "floating-point 32" and the "fp32".

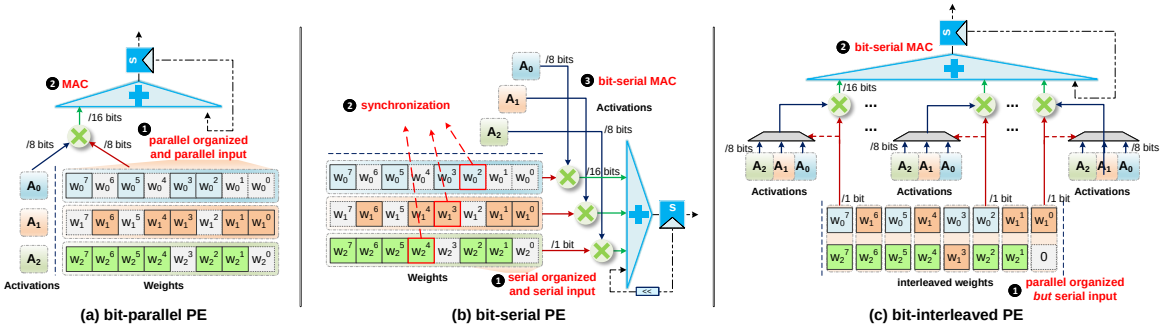


Fig. 1. Examples for the bit-interleaved PE comparing with prior bit-parallel/serial PE, where the w_i^j marked in grey is the non-essential bit (0 bit). (a) bit-parallel PE. Step ❶ organizes the weights for MAC in parallel; Step ❷ issues MAC. (b) bit-serial PE. Step ❶ organizes the weights in serial; Step ❷ synchronizes the significance of the essential bits; Step ❸ issues the "bit-serial" MAC. (c) bit-interleaved PE. Step ❶ organizes the weights in parallel, but Step ❷ issues the bit-serial MAC along each bit significance, excluding the synchronization operation.

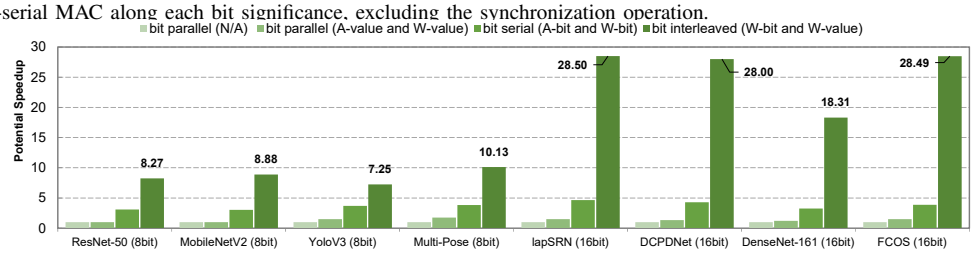


Fig. 2. Potential speedup of bit interleaving. Most existing sparsity-aware accelerators only support fixed-point precision, so we only compare 8b and 16b DNNs in Table II. The floating-point applications will be evaluated over GPUs in Section V.

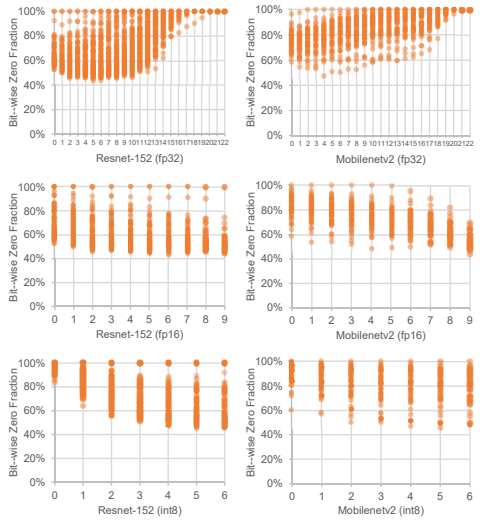


Fig. 3. Sparsity parallelism. The X-axis indicates the bit significance of the mantissa, and each dot indicates the fraction of zeros on this bit lane across all the weights of this kernel. It shows $\sim 50\%$ bits are 0s for all kernels. On X-axis in the figure, the sparsity only entails the mantissa (23/10 bits for float 32/16), and 7 significant bits excluding the sign bit for int8 precision.

W-value in Table I) for various AI tasks. In addition, mining the activation sparsity is also workable, which depends on the data reuse policy.

Ideally, an accelerator should be applicable to most cases, providing enough convenience and flexibility for the end-users in synergy. However, the vast majority of accelerators can only support fixed-point inference, which makes it difficult to work from a general-purpose perspective. For instance, the back propagation in DNNs training relies on floating-point precision to ensure the training accuracy. In this work, due to the bit interleaving design concept, *Bitlet* could not only handle sparsity effectively but also support versatile precisions including both fixed point and floating point, which renders it suitable for both high-performance and power-efficient scenarios.

B. Leveraging the Sparsity Parallelism

Previous researches have proved that bit-level sparsity is abundant. However, they only focus on exploring the strategy of skipping zero bits inside a particular weight, while none of them investigate the inter-weight sparsity, that is, the *sparsity parallelism* based on which bit interleaving may dramatically outperform the bit serial/parallel prototypes.

As shown in Figure 3, we trace the bit sparsity for different convolution kernels and find that the weight sparsity at each significance is uniform. For the two DNNs `ResNet152` and `MobileNetV2`, the first half of the mantissa (bit0∼bit16) shows obvious aggregation, which means the amount of 0s and 1s that lie on this bit significance is nearly comparable. This provides an opportunity to load the weights in parallel but compute the product in serial along each bit lane. The independence of bit lanes helps to avoid synchronization. As will be shown in Section V, the computation of 64 MACs could be finished within one cycle in our FPGA platform for most of the evaluated DNNs.

Besides, from bit lane 17∼23, the dots mostly overlap at 100% on the Y-axis (the long tail in the fp32 figures), which indicates most of the bits are 0. The floating-point multiplier does not distinguish this sub-optimal case, because it is designed for covering any corner case of the operand, which is also the fundamental reason that floating-point MAC can hardly be accelerated. Whereas in our proposed *Bitlet* accelerator, such abundant sparsity could be easily exploited by bit interleaving. The details will be discussed in the following section.

III. Bit Interleaving

A. Theorem

Uniformly a floating-point operand can be splitted into 3 parts: signed bit (S), mantissa (M), and exponent (E), following IEEE-754 [12]. We demonstrate the scenario of the

float-32 format (fp32 hereafter) computation first.The mantissa contains 23 bits and the exponent occupies 8 bits with the last bit for the sign. A floating-point operand fp could be expressed as $fp = (-1)^s 1.m \times 2^{e-127}$, in which e is the actual position of the "binary point" plus 127. We consider the scheme that a series of fp32 MACs to compute one partial sum:

$$\sum_{i=0}^{N-1} A_i \times W_i = \sum_{i=0}^{N-1} (-1)^{S_{W_i}} A_i \times M_{W_i} \times 2^{E_{W_i}} \quad (1)$$

Explain Eq.1 first: we translate W_i into the normalized fp32 representation, in which M_{W_i} and E_{W_i} stands for $1.m_{W_i}$ and $e_{W_i} - 127$ for simplified expressions. Note that, the M_{W_i} is the fixed-width mantissa (24 bits), which includes the hidden bit – the first bit '1' in the mantissa compiled with IEEE-754 format. M_{W_i} can be further decomposed to get the bit-represented partial sum:

$$\sum_{i=0}^{N-1} A_i \times W_i = \sum_{i=0}^{N-1} \sum_{b=0}^{-23} \left[(-1)^{S_{W_i}} A_i\right] \times 2^{E_{W_i}+b} \times M_{W_i}^b \quad (2)$$

$$= \sum_{i=0}^{N-1} \sum_{b=0}^{-23} \left[(-1)^{S_{W_i} \oplus S_{A_i}} \cdot M_{A_i}\right] \times 2^{E_{W_i}+E_{A_i}+b} \times M_{W_i}^b \quad (3)$$

where $M_{W_i}^b$ is the b-th bit of the binarized M_{W_i}. Replacing A_i with IEEE-754 binary format, Eq.2 could be rewritten as Eq.3. Furthermore, let $E_i = E_{W_i} + E_{A_i}$, then we can get:

$$\sum_{i=0}^{N-1} \sum_{b=0}^{-23} \left[(-1)^{S_{W_i} \oplus S_{A_i}} \cdot M_{A_i}\right] \times 2^{E_i - E_{max}} \times 2^{E_{max}+b} M_{W_i}^b \quad (4)$$

$$= \sum_{i=0}^{N-1} \sum_{b=E_i-E_{max}}^{E_i-E_{max}-23} \left[(-1)^{S_{W_i} \oplus S_{A_i}} \cdot \left(M_{A_i} \times M_{W_i}^b\right)\right] \times 2^{E_{max}+b} \quad (5)$$

In Eq.5, we can infer that N number of fp32 MACs is equivalent to a series of bit-level operations of the corresponding mantissa. In specific, if $M_{W_i}^b = 1$, then the summation of N MACs is transformed into the summation of N signed M_{A_i}(denoted by $(-1)^{S_{W_i} \oplus S_{A_i}}$) shifting $2^{E_{max}+b}$, contingent on the significance bit b and the E_{max}.

We can conclude a fact from the above analysis that the floating-point partial sum could be turned into bit-level operations, with the sparsity which could be exploited at the bit-level. The product is primarily formed by the mantissa M_{A_i}, but whether it would actually contribute to the product is decided by the single bit in corresponding weight $M_{W_i}^b$ in Eq.5. The above bit sparsity could be also exploited in bit interleaving. According to the previous sections, each bit significance of weight in existing DNN models is composed of a significant portion of 0s, so if $M_{W_i}^b = 0$ but another weight W_j at the same significance b is an essential bit 1, we can allow $M_{W_j}^b$ taking the place of $M_{W_i}^b$, making different weight bits interleaved in the same lane. In a hardware implementation, it means the mantissa M_{A_j} and M_{A_i} could be contributed to the final product in tandem accelerating the computations by exploiting the sparsity.

The theorem also fits the fixed-point precision. In Eq.5, the impact of E_{max} and $E_i - E_{max}$ could be skipped for fixed-point numbers, because fixed-point values don't have an exponent part. In other words, we just ignore the exponent part in the equation and the others remain the same for the calculation of the fixed-point numbers. Next, we describe how bit interleaving can work for fp32 format, and in the next section, we elaborate on the architecture of the proposed *Bitlet* accelerator and how to support versatile precisions with the concept of bit interleaving.

B. Procedures

Figure 1c shows the computation step of bit interleaving for the fixed-point MAC of 8-bit. But the MAC for floating-point is not easy to be harnessed as its fixed-point counterpart, because there is an exponent part residing in the binary encoding, and different operands' exponents usually differ. To exploit the maximum potential of the sparsity, bit interleaving shows three essential steps based on Eq. 5:

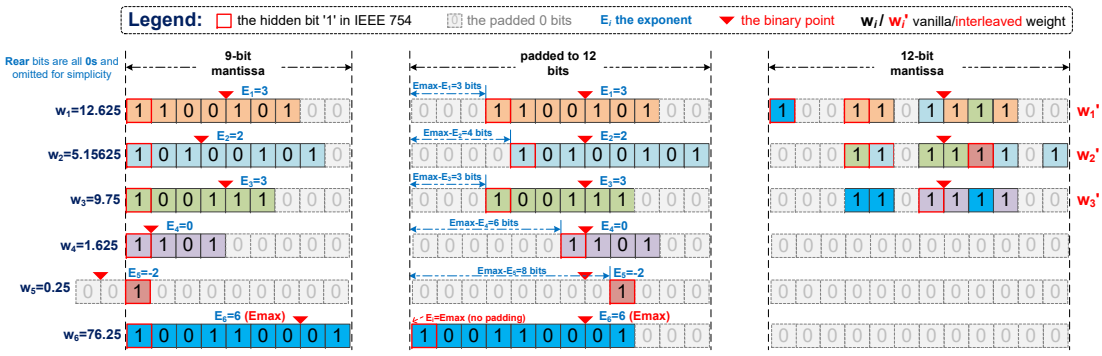
1) Step ❶: Pre-Processing: Figure 4a illustrates as an example of 6 fp32 weights arranged in rows, each with an arbitrary exponent and mantissa. The triangle mark shows the real location of the binary point. For simplicity, we use the more iconic representations to express the value. For example, the $(0.01)_2$ with $E_5 = -2$ hence stands for 0.25 in decimal (W_5). This step is similar to Step ❶ in Figure 1c. The only difference here is that this step organizes the fp32 weights in parallel for interleaving. It pre-processes these binary weights so as to obtain the exponent separately and then further identifies the "maximum" exponent value (E_6 in this example). The mantissa is also interpreted and stored for the later MAC operation. For readability, the rear bits (bit $9 \sim 23$) of each mantissa is omitted.

2) Step ❷: Dynamic Exponent Matching: The exponent part for a float-point number actually denotes the position of the binary point. Conventionally in hardware implementation of float-point calculations, it involves a step called "exponent matching" to align the binary point of the operands. In bit interleaving , we match the exponents according to the maximum value (E_6 in this example) of all exponents to align a group of floats, instead of handling them in one pair and then another. This step is called "dynamic exponent matching".

Recalling Eq.5, for hardware implementation, the two $\sum$s could be executed parallelly: the outmost $\sum$ represent the vertical dimension in Figure 4a, that is, N number of weights with their associate activations; the innermost $\sum$ shows the horizontal dimension which represents different bit widths of the mantissa. Therefore, the key concept of Eq. 5 is to compute all the M_{A_i}s with $M_{W_i}^b = 1$, along the two dimensions in Figure 4a.

The convolution operation, that we aim to accelerate, is to compute $\sum_{i=0}^{N-1} A_i \times W_i$, and it involves N number of activation and weight MACs. Rather than an costly one-by-one match, this step is intended to match all exponents to the maximum each time. It can be seen in Figure 4b that, the 6 weights are all aligned to the maximum exponent–w_6. For instance, w_5 needs to shift 8-bit positions to the right to align with w_6. As an advantage, the exponent matching step is only issued once in computing one parital sum which resulting in an efficient hardware implementation.

3) Step ❸: Essential Bit Distillation: Now is to take advantage of the essential bits to obtain the proper result



(a) **Step 1:** preprocessing the floating-point weights

(b) **Step 2:** dynamic exponent matching

(c) **Step 3:** bit distillation

Fig. 4. Core concept of "*bit interleaving*". The fp32 weights are exemplified and pre-processed in Step ❶. The weights are shifted and zero-padded according to the maximum exponent (E_{max}) in Step ❷. Note that exponential matching to the right-hand side is only allowed in the case of severe accuracy loss as specified in the IEEE 754 standard. Step ❸ handles bit distillation. Final 3 weights that are interleaved finally participated in the computation step in the accelerator.

of MACs and further, the satisfying inference speed. As the sparsity parallelism mentioned in Section II, this step distills the essential bits, which is exactly identical to the Step ❷ in Figure 1c.

As shown in Figure 4c, the computations could be totally reduced significantly from 6-operand MACs to only 3 by the effective distillation. Still using W_6 for the example, its exponent is 6 and the first bit (b=0) is an essential 1. Inspired by the formula Eq.5, the value of $2^{E_{max}+b}$ for this bit equals 2^6, which means this bit lies in the $7th$ position before the binary point. For $W_1 \sim W_5$, the 2^6 positions are all zeros after the exponent matching step. If we ascend the first bit of W_6 replacing the same position in W_1 in the same vertical lane, we are able to compute $A_6 \times 2^6 + A_1 \times 2^3$ simultaneously. The essential bits in other weights could be operated the same way, and the distilled weights are finalized in Figure 4c.

As a summary, the above steps of the bit-interleaving method accelerate fp32 MACs from two aspects: first, it eliminates the conventional one-by-one expensive exponent matching operations by introducing the multipair exponent dynamic matching mechanism and second it eliminates the inefficient computations caused by unnecessary bits by benefiting from the parallel bit-level sparsity. In Section V, we will demonstrate that on our accelerator platform, the fp32 MAC can compute even faster than int8 or fixed-16 quantization. Moreover, it could accelerate the original DNNs in various precisions and support the other software algorithm in general-purpose usage as well without hardware change.

IV. Bitlet Accelerators

To implement the bit-interleaving method in hardware, we design a novel accelerator couple, including two members named *Bitlet* and *Bitlet-X* respectively. In this section, we firstly demonstrate the common key micro-architecture with versatile precision support in both of them, as well as the *Bitlet-X* specific micro-architecture, which optimizes power and the inference speed compared with *Bitlet*. An efficient memory system is also used for constructing the overall accelerator.

A. "Bitlet" Compute Engine

Key Module #1 - Preprocess. Firstly, we design the *preprocessing module* in order to accomplish the first two steps of the bit interleaving theorem mentioned previously. For simplicity, We define N as the maximum number of A/W pairs that *Bitlet* is designed to handle as in Figure 5. In *Bitlet* Compute Engine (*BCE* hereafter), $A_0 \sim A_{N-1}$ are the activations, while W_0 through W_{N-1} are the original weights which are needed to be multiplied correspondingly. Proposed *preprocessing module* decomposes each W_i and A_i pair into two parts: mantissa(including the sign information) and exponent, and then sums up the E_{W_i} and E_{A_i} as the value E_i for each A/W pair. A Compare Tree is designed to find the maximum value(E_{max}) among all E_i. The E_{max} is then stored in the register and stays the same during the subsequent dynamic matching phase. After keep the value of E_{max}, M_{W_i} is shifted by $E_{max} - E_{i_i}$ bits to align its exponent to E_{max}. Recalling the Figure 4, E_{max} should be $E_6 = 6$ in W_6, other weights are all aligned to E_6, *i.e.,* M_{W_4} will be shifted by $6 - 0 = 6$ positions to the right-hand side as shown Figure 5. The left shift position is automatically padded with 0s, and any subsequent bits beyond $b = 23$ are discarded because the mantissa is 24 bits long.

Key Module #2 - Wire Orchestrator. After the dynamic matching phase, a 24-bit shifted mantissa is obtained, which is indicated by $M_{W_i}[0] \sim M_{W_i}[23]$. Then they are processed by *Wire Orchestrator* in Figure 5. According to the arithmetic representation mentioned previously, the bits that share the same bit significance in mantissa should be processed sequentially later. So proposed *Wire Orchestrator* is used to reorganize the preprocessed mantissa and aggregate the same bit significance together from different pairs of mantissa. Particularly the output of the *Orchestrator* is represented as $M_{W_0}[b],\ M_{W_1}[b],\ \cdots,\ M_{W_{N-1}}[b]$ in which b is in range $0 \sim 23$. This module contains no combinatorial or sequential logic because it only assembles wires for the aligned mantissa in the previous pre-processing step and performs the *transpose* operation, so intuitively this module only introduces tiny power consumption but to some extent increase the circuit area. The actual influence of this module will be evaluated in later Section V.

Key Module #3 - RR-reg. The proposed *RR-reg*$_i$ is designed to distill the essence bit in the interleaved values and select one output of *BCE* from the N activation mantissa. As

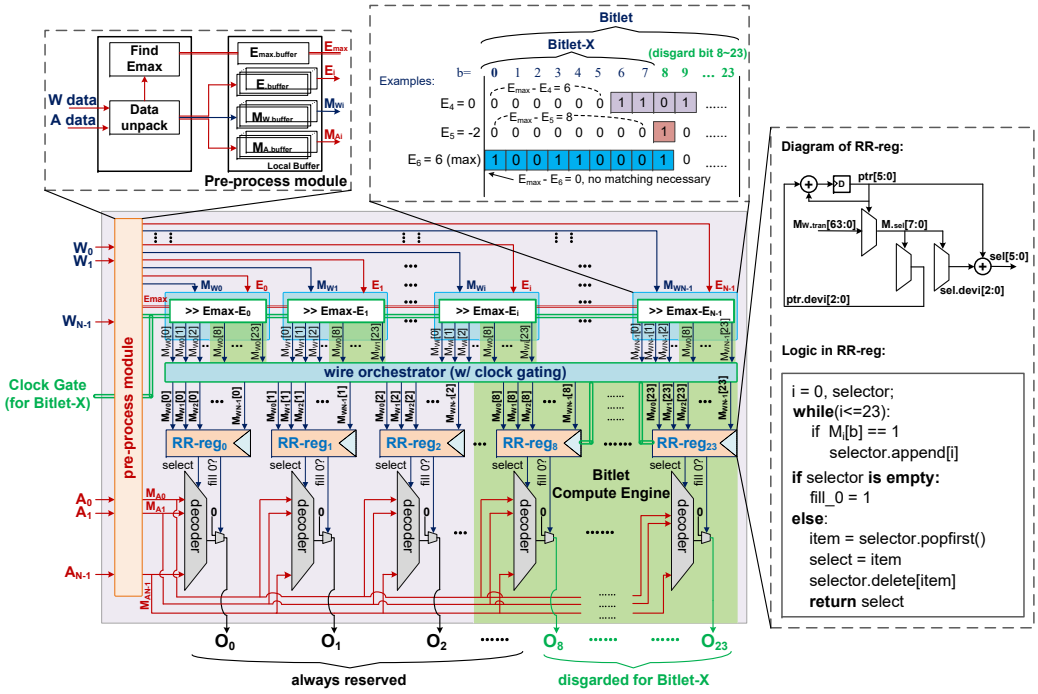


Fig. 5. Microarchitecture of the core module – *Bitlet Compute Engine (BCE)*. The green area contains the modules that could be clock gated for *Bitlet-X*.

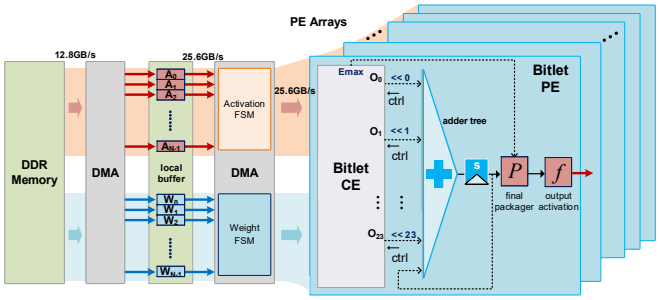


Fig. 6. *Bitlet* accelerator. To compute the partial sum, each *Bitlet PE* is composed of one *BCE* and a adder tree. *Bitlet* is versatile: for the floating point, Emax is dynamic, while for the fixed-point precisions, Emax is fixed to the target precision (*i.e.,* 16 or 8).

shown in Figure 5, the pseudo-code shows how *RR-reg* works: it concentrates the input bits of the same bit significance from the *orchestrator* and distills one essential bit each cycle in sequence. The "select" signal could inform the decoder logic to configure the chosen activation path and output O_i, which would be accumulated with other outputs later. *RR-reg* activates the "fill 0" signal when there is no essential bit detected, and the output O_i will be 0 as well. This "filling 0s" operation is used to avoid the scheme of the input for one *RR-reg* are 0s, *i.e.,* $b = 1$ or 2 in Figure 4c.

Discussions: We highlight three advantages of the *BCE*: ① The proposed architecture does not influence the inference accuracy, due to the reason that the dynamic exponent matching mechanism mentioned in Section III-B2 is coincident with the float-point algorithm in IEEE 754, which is to discard the rightmost outlier bits after shifting ("$>> E_{max} - E_i$" in Figure 5). These bits are insignificant in the calculation and cause almost no effect, hence the result will not be changed using this method. ② Unlike other sparsity-aware accelerators, our proposed *BCE* doesn't need an extra and costly sparsity-aware pre-processing method to finish its computation. Since the pre-processing module in Figure 5 is just used to distinguish the mantissa/exponent part of each A/W pair. In our circuit implementation, we instantiate a sliding window with limited window size in each *RR-reg* to distill the essential bits to avoid large area consumption. Benefiting from the sparsity parallelism, the distillation of the essential $M_{W_i}[b]$ could be finished nearly in tandem in each *RR-reg*. Since it is the fundamental module in the whole hardware design, the *Bitlet* accelerator is with promising throughput and low area cost as demonstrated in Section V. ③ Except for the *RR-reg*, the *BCE* consists almost of combinatorial circuits, but it does not involve complex wires that could lead to extended critical path delays. Each *RR-reg* spawns one output O_i per clock cycle, but from the system level's perspective, the cycle time spent on one partial sum is greatly optimized compared to the traditional one-to-one MAC in our later evaluation. N is the key design parameter in *BCE*, and larger N has the ability to distill more bit 1s when computing one partial sum. In Section V we will also study the $N's$ influence on the inference speed and explain how we choose the best N for the hardware design.

B. "Bitlet-X" Compute Engine

As previously mentioned, *Bitlet* is an accelerator couple. Besides the above general-purpose *BCE* micro-architecture to empower training and inference in one platform, an additional prototype for inference-only optimization - *Bitlet-X* is also proposed by updating the *BCE* micro-architecture.

As shown in Figure 5, the major modification for *Bitlet-X* resides in the wire interconnections within the *BCE*. It instantiates a series of clock gating signals that traverse from each shifting module ("$>> E_{max} - E_i$") to the wire *orchestrator* and finally some of the *RR-regs*. The clock-gated wires for saving power are marked as green in the figure. For example,

Bitlet-X only reserves the most significant 8 bits in each weight mantissa for shifting, and the wires of bit $M_{W_i}[8] \sim M_{W_i}[23]$ are clocked gated. Similarly, *RR-reg*s[8] through *RR-reg*s[23] are also clock gated, only keeping the front 8 *RR-regs* alive for the essential bit distillation. The final product only accounts for $O_0 \sim O_7$, so the $O_8 \sim O_{23}$ could be kept idle and safely deactivated to save power.

The core basis of this inference-only update stems from the feature of deep neural networks, that is, the precision of weight/activation operands can be compromised to get faster inference speed in return, and meanwhile, the accuracy of the network is also able to be maintained. The infrastructure of the *BCE* naturally provides such an opportunity to approximately compute the partial product but keep the final product intact. In specific, the shifting module is responsible for shifting the less significant bits, based on the quantitative relationship between E_{max} and E_i, to the right-hand side and truncating the bits beyond the certain range, i.e. for *Bitlet* we choose 23 *RR-regs* while for *Bitlet-X* we choose only 8. The truncated bits are undoubtedly harmless to the final accuracy because they are too tiny in value. More convincingly, we have proved in Section V-D that even more aggressive *Bitlet-X* could achieve nearly identical performance on our benchmark suite containing various famous deep learning tasks and the associate evaluating metrics.

Besides, the merits of the manipulation are also twofold: (1) As an approximation optimization, *Bitlet-X* gracefully reduces the wiring complexity, no matter for the input pre-processing module, shifting, or for the *wire orchestrator* and *RR-regs*, which means the power and area could also be optimized accordingly (proved in Section V-H). The benefit further spreads to the simplified thermal management and shortened critical-path delay. (2) Due to the truncation of the mantissa, the input operand only has 8 bits that really matter instead of the original 24 bits in the standard float-32 mantissa. Therefore, we could store lesser bits in the on-chip SRAM, from the original 32 bits (1-bit sign, 8-bit exponent, 23-bit mantissa) to the current 16 bits (1-bit sign, 8-bit exponent, 7-bit mantissa, note that the first '1' in the mantissa is always hidden), because *Bitlet-X* only takes the most significant 8 bits into account. This benefit leads to the storage reduction by 50%, or in other words, acquires $1\times$ extra throughput improvement. Finally, it is worth mentioning that *Bitlet-X* cannot be used for training because such an approximation method does bring precision variation, so the backpropagation during training is also affected by unpredictable gradient descent. It is designed for boosting the inference performance only (see Section V-C).

C. Accelerator Architecture

PEs. To build an accelerator with large throughput, we call the combination of one *BCE* and the adder tree with other post-processing logic as a PE. Each PE receives the operands (weights and activations) simultaneously according to the entire dataflow and spawns the output O_i of different bit significance being the input of the adder tree. PEs are connected and compose the whole *Bitlet* accelerator, as shown in Figure 6.

It finalizes the result by multiplying $2^{E_{max}+b}$ for the correctness of the final result. It can be decomposed into a fixed part b and a common part E_{max} for *BCE* output. The fixed part of the exponent is calculated by shifting the fixed amount of the wire connection operation in the physical circuit. As regards the common component, E_{max} is then used to generate a formatted result by applying the accumulator result in the final packing module. Obtaining O_i only requires fixed-point addition, instead of any multiplication, which also reduces arithmetic complexity and power consumption.

Memory System. To improve the throughput, the *Bitlet* accelerator provides separated DMA channels for the activation and weight data. As shown in Figure 6, the data fetched from DDR3 memory will be stored in the local buffer thus providing adequate bandwidth for the accesses from the corresponding *Bitlet PEs*. In the RTL implementation, the bandwidth could achieve 12.8 GB/s per channel between the memory and the local buffer, and the accelerator utilizes a total of 25.6 GB/s to fetch the activation and weight data from the buffer. Considering dataflow, *Bitlet* utilizes WS(weight stationary) dataflow and activation broadcasting method [11] to minimize the main memory accesses.

D. Versatility

The proposed 2 *Bitlet* accelerators is a versatile accelerator couple, which can be manifested in two aspects:

For the *Bitlet* prototype, it better balances the "generality-and-efficiency" trade off. It could be conveniently configured into the fixed-point mode, showing enough flexibility in the end scheme. For FXP16(fixed point 16-bit) precision, we could easily just power gate the processing module of the function part that performs the exponent matching and shifting ("$\gg E_{max} - E_{W_i}$" in Figure 5), and the input W_i can be directly connected to the *Wire Orchestrator*. *Bitlet* is designed for the 24-bit mantissa to support the multi-precision, in the case when we just use FXP16 format, only *RR-reg*$_0 \sim$ *RR-reg*$_{15}$ are involved in the practical computation. Other *RR-reg* could be safely powered gated or left idle in static with trivial power consumption. Similarly for int8 quantization and other precision (*i.e.,* int4, int9 *etc.*), *Bitlet* could handle it in the same manner. Benefitting from this, the end users on our platform do not have to change to use other precision-specific accelerators due to different precision demands. They are free to calibrate their DNNs on our platform to achieve accuracy goals and power/performance tradeoffs.

For the *Bitlet-X* prototype, it tackles the "speed-and-accuracy" trade off (for inference only). By precisely targeting the front 8 most significant bits in the mantissa, it achieves even faster yet more accurate inference for the DNN models. The two members of the couple are easily configurable. The major components are highly identical between the couple designs except for the clocking gating signals in *Bitlet-X*. The end users could balance the performance, power, or efficiency of the training and inference procedure in a more flexible way without much effort.

V. Evaluation

In this section, our presented bit interleaving scheme and the *Bitlet* accelerator will be evaluated.

TABLE II
DNNs BENCHMARK.

Networks	Type	Precision	Domain	Dataset	GFLOPS	Weights	W-bit Sparsity (%)
MobileNetV2 [28]	2D Convolution	8 bit	Image Classification	ILSVRC'12 [3]	0.615	3.49M	76.85 (fixed point)
YoloV3 [27]	2D Convolution	8 bit	Object Detection	CoCo [1]	25.42	61.95M	77.78 (fixed point)
ResNet-50 [14]	2D Convolution	8 bit	Image Classification	ILSVRC'12 [3]	8.21	25.56M	70.15 (fixed point)
Multi-Pose [19]	2D Convolution	8 bit	Pose Estimation	CoCo [1]	97.55	59.59M	66.33 (fixed point)
DCPDNet [38]	Encoder -Decoder	16 bit	Deraining /Dehazing	NYU-Depth [31]	254.37	66.9M	75.00 (fixed point)
DenseNet-161 [15]	2D Convolution	16 bit	Image Classification	ILSVRC'12 [3]	15.56	28.68M	68.92 (fixed point)
lapSRN [20]	2D De-Convolution	16 bit	Image Super Resolution	SET14 [4]	736.73	0.87M	74.31 (fixed point)
FCOS [32]	Feature Pyramid	16 bit	Object Detection	CoCo [1]	80.14	32.02M	70.83 (fixed point)
Transformer [34]	Seq2Seq	float 32	Natural Language Processing	wmt'14 [6]	10.6	176M	45.75 (floating point)
C3D [33]	3D Convolution	float 32	Video Understanding	UCF101 [5]	38.57	78.41M	45.83 (floating point)
CartoonGAN [9]	GAN	float 32	Style Transfer	flickr [2]	108.98	11.69M	48.49 (floating point)
D3DNet [37]	3D Deformable	float 32	Video Super Resolution	Vimeo-90k [35]	408.82	2.58M	47.69 (floating point)

TABLE III
ACCELERATORS BENCHMARK.

	Chip	PEs/Core	Precision	Tech. (@TSMC)	Freq. (MHz)	PEAK Performance (GOPs)	Power	PEAK Power Efficiency (GOPs/W)	Area (mm^2)
Accelerator ASICs	Eyeriss [11]	168	16b	65nm	250	84	278mW	83.09	12.25
	SCNN [26]	64	16b	16nm	1000	2000	-	-	7.9
	Stripes [17]	4096	$1 \sim 16b$	65nm	980	-	-	-	122.1
	Laconic [29]	192	$1 \sim 16b$	65nm	1000	-	-	441 (16b), 805 (8b)	1.59
	Bitlet	**32**	**fp32/16, 1∼24b**	**28nm**	**1000**	**204.8 (fp32) 372.35 (16b) 744.7 (8b)**	**570mW (fp32) 432mW (16b) 366mW (8b)**	**359.15 (fp32) 667.97 (16b) 1335.93 (8b)**	**1.54**
				65nm			**1829mW (fp32) 1390mW (16b) 1199mW (8b**	**111.97 (fp32) 267.87 (16b) 621.10 (8b)**	**5.80**
	Bitlet-X	**32**	**fp32/16, 1∼24b**	**28nm**	**1000**	**204.8 (fp32) 372.35 (16b) 744.7 (8b)**	**438mW (fp32) 432mW (16b) 366mW (8b)**	**467.58 (fp32) 667.97 (16b) 1335.93 (8b)**	**1.54**
				65nm			**1448mW (fp32) 1390mW (16b) 1199mW (8b**	**141.44 (fp32) 267.87 (16b) 621.10 (8b)**	**5.80**
GPUs	Titan V	5120	fp32/16, 8b	12nm	1455	14900 (fp32) 29800 (fp16)	250W	59.6 (fp32) 119.2 (fp16)	-
	Titan Xp	3840	fp32, 8b	16nm	1582	12150 (fp32)	250W	48.6 (fp32)	-
	Tegra X2	256	fp32/16	16nm	854	750.1 (fp32) 1330 (fp16)	15W	50.0 (fp32) 88.7 (fp16)	-

A. Experimental Setup

DNN Models & Application Baselines. *Bitlet* is a general-purpose accelerator which can support $1 \sim 24$bit fixed-point and floating-point precision. As listed in Table II, we select 12 DNNs applications with different network structures as DNNs benchmarks. As for hardware comparison, we choose a series of baselines: (1) *GPUs*, containing high-performance data center devices – Titan V (volta) and Titan Xp (pascal), and power-efficient edge-level device – Jetson TX2 (pascal). (2) *Sparsity-agnostic accelerators*, typical accelerator based on the fixed-point MAC - Eyeriss [11], which utilizes multipliers and adders both designed for fixed-16 values without taking into account the sparsity; typical bit-serial accelerators - Stripes [17]. (3) *Sparsity-aware accelerators*, Laconic [29] and SCNN [26] are adopted for its bit-sparsity and value-sparsity design. For *Bitlet*, we evaluate its flexible precisions, including 8-bit, 16-bit, and float 32. We select a number of common configurations $(2, 4, 8, 16, 32, 64)$ for the crucial design parameter N to assess its effect on the speedup. In addition, in order to test the performance sensitivity of different modules in *Bitlet*, we carry out a *hardware ablation study*. To get the best performance, we set N=64 as the default configuration in all experiments.

FPGA & ASIC implementation. We first complete the prototype verification based on Xilinx Virtex-7 Series FPGA. Under the frequency of 200MHz, we equipped the *Bitlet* with 32 PEs. Based on the above configurations, post-synthesis simulation is executed using the Vivado platform (v2018.2). We record the runtime memory access data and send it to the DRAMsys tool [18] to evaluate the energy consumption. Moreover, at each run, the inference time is recorded in *frames*

TABLE IV
QUANTITATIVE COMPARISON OF AVERAGE COMPUTATION PERFORMANCE (CYCLES/64-MACS).

	16b				8b			
	DenseNet-161	lapSRN	FCOS	DCPDNet	Multi-Pose	ResNet-50	YoloV3	MobileNetV2
Eyeriss	343.68	339.34	346.62	357.39	341.71	344.08	345.08	343.77
SCNN	281.87	226.22	231.08	262.09	197.14	344.08	230.05	343.77
Stripes	150.86	90.49	115.54	106.25	110.23	158.39	119.94	174.07
Laconic	105.18	71.44	89.77	83.64	89.92	111.13	93.22	113.44
Bitlet	22.85	29.51	22.44	16.45	24.64	21.03	19.04	22.22

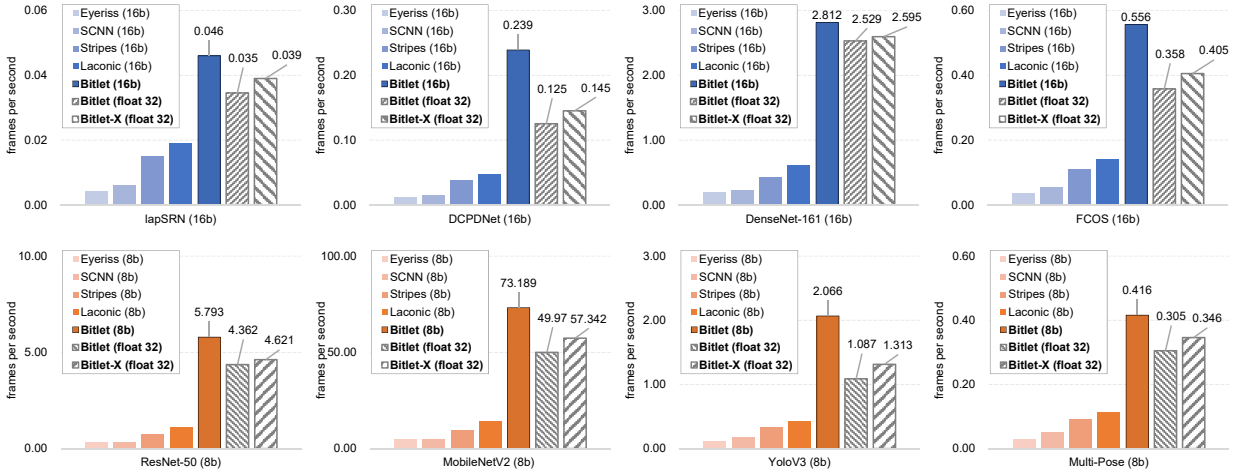


Fig. 7. Speedup comparison. The upper row and bottom row indicate the 16b and 8b DNN benchmarks, respectively. For reference, we also run the float-32 version on *Bitlet*. All of the results are actual frames per second (fps) values, with more being better.

per second (fps). For ASIC, we set the frequency to 1 GHz, and estimate the power and area under the TSMC $28nm$ and $65nm$ process technology based on Synopsys Design Compiler (v2016). To record a detailed module-level area, we decompose the *Bitlet PE*. The baseline data of the area are directly provided from their publications, however, only the total PE area is available. Note that the data for our implementation is from the pre-layout (or netlist level), while the baselines use the post-layout estimation.

B. Specifics Comparison

We compare the specs of the SOTA GPUs and accelerator prototypes with the *Bitlet* couple. For the "peak performance", we implement the *Bitlet&Bitlet-X* with Verilog and directly collect the data from the literature for the baseline accelerators. For some of the data not mentioned in the literature, we use a dash '-' instead in Table III. Under the case of 32 PEs/BCEs, *Bitlet* achieves a 744.7 GOPs, 372.35 GOPs, 204.8 GOPs peak performance and 1335.93 GOPs/W, 667.97 GOPs/W, 359.15 GOPs/W peak efficiency under 28 nm process technology (621.10 GOPs/W, 267.87 GOPs/W, 111.97 GOPs/W under 65nm process technology) for the fixed-point 8b, 16b and floating-point 32 precision, respectively. *Bitlet-X* updates its *BCE* with clock gating for faster and power-efficient inference under floating-point 32 precision. The peak performance in GOPs remains the same, but the "peak power efficiency" is effectively boosted to 467.58 GOPs/W, $1.3\times$ higher than the vanilla *Bitlet* under 28 nm technology. As indicated in Table III, Titan V achieves the maximum performance, but its efficiency is compromised by high power. Also, despite having the fewest PEs, *Bitlet* achieves the best efficiency and area under the $28nm$ technology node, where the "PEs/Cores" is the standard configuration, based on which the peak power efficiency and performance are reported. To be fair, the same number of PEs are employed with approximately comparable processing resources for performance and energy evaluation.

C. Speedup and Energy Efficiency

Speedup. As shown in Figure 7, we use frames per second (fps) measured on the FPGA to compare the speedup. For instance, *Bitlet* can achieve 2.8 fps when running DenseNet-161. However, the results are only 0.611 ($4.6\times$), 0.426 ($6.6\times$), 0.228 ($12.33\times$) and 0.187 ($15.03\times$) on other accelerator baselines. Among them, Stripes, as the bit-serial accelerator, enables layer-wise configurable precision from $1 \sim 16b$, verified offline based on the least acceptable accuracy loss. SCNN fixes the precision to 16b in our evaluation and highly relies on the value sparsity. However, pruning is not implemented on our benchmarks, therefore it performs worse than Stripes.

From Figure 7, we discover that the fps result of *Bitlet* and *Bitlet-X* in float-32 precision is even faster than all the fixed-point baselines, where the bit-interleaving scheme causes the reason. By distilling the sparsity in parallel rather than in serial, the floating-point MAC also gains acceleration by taking advantage of the bit-level sparsity. The fact that the DNNs may directly obtain plentiful acceleration on *Bitlet*, without the labor-intensive and time-consuming quantization operations, is hence a more significant finding.

Table IV shows the performance of carrying out one MAC operation. *Bitlet* represents $22 \sim 29$ cycles/MAC between 16b and 8b, and the 16b *Bitlet* acts almost comparably to the 8b *Bitlet*. That makes sense since, despite variations in bit length, bit sparsity displays essentially uniform distribution at each significance. This feature demonstrates that *Bitlet* may act as a general-purpose accelerator and support any fixed-point precision.

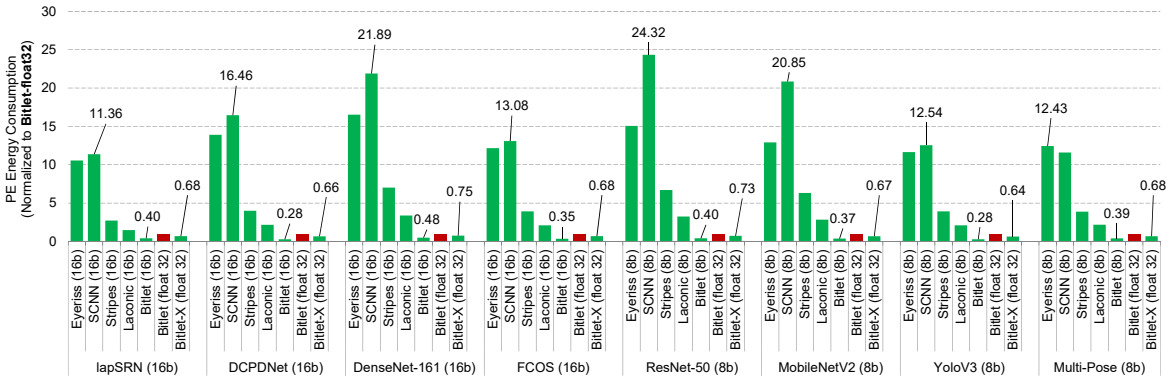
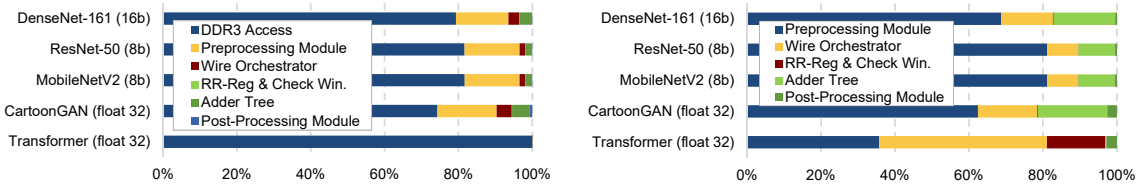


Fig. 8. Energy consumption. Also, we present the float-32 inference's energy result, and all the fixed-point results are normalized to it.



(a) Full system energy breakdown. (b) PE-only energy breakdown.

Fig. 9. Energy breakdown.

Energy Consumption. Figure 8 demonstrates the energy consumption, which is normalized to the "*Bitlet* (float 32)". The biggest difference is shown at ResNet-50, which consumes $24.31\times$ more energy consumption. From the experiment data, SCNN does not perform well. Due to the value-level sparsity methodology in SCNN, the speedup is only obvious when the sparsity of the value level is abundant. However, the 12 applications do not go through sparse training or pruning. Therefore, the accelerator designed for indexing the sparsity for zero skipping cannot fully contribute to the speedup improvement but still consumes a significant amount of power. *Bitlet* (16b) has the lowest energy consumption. For *Bitlet-X*, the energy consumption is smaller than the vanilla *Bitlet* due to the clock gating of most of the wires and *RR-regs* in the *BCE*, and that leads to decreased power consumption during inference.

Energy Breakdown. Figure 9 shows the energy breakdown in two aspects. As shown in Figure 9a, it presents the full-system energy breakdown and the memory accesses dominate the energy consumption. Especially for Transformer, the data attains almost 99% while the PE computation energy uses only 1%. As shown in Figure 9b, the PE-only energy for each DNN is further decomposed, which shows the domination of the preprocessing module (63.2%). This is because it involves a large number of buffers for the mantissa and the exponent. For other modules, *Wire Orchestrator* and adder tree consume 14.7% and 6.27% energy on average, respectively.

Efficiency. *Bitlet* can support the $1 \sim 24$b fixed-point and floating-point precisions. Since the accelerator baselines can't support the floating-point arithmetic, we compare the power efficiency of *Bitlet* with the GPU baselines. Table V shows training and inference efficiency data, and *Bitlet* shows $81.16\times$, $4.72\times$, $1.80\times$ and $34.81\times$ improvement over the Titan V. Correspondingly, the inference efficiency improvement is $12.9\times$, $15.29\times$, $3.21\times$ and $21.09\times$. We can find an obvious efficiency gap between the training and inference, where the reason is that the backward propagation in the training can not be accelerated by *Bitlet*.

TABLE V
TRAINING/INFERENCE EFFICIENCY IN THE FLOAT-POINT PRECISION. NOTE THAT TX2 IS NOT USED FOR TRAINING.

GPU Baselines	Models (**Train / Inference** efficiency in GOPs/W)			
	Cartoon -GAN	Trans -former	C3D	D3DNet
Titan V	0.27 / 3.19	4.09 / 21.80	1.82 / 4.32	0.06 / 4.67
Titan Xp	0.20 / 2.36	3.03 / 16.13	1.35 / 3.19	0.04 / 3.46
Jetson TX2	– – / 0.20	– – / 1.39	– – / 0.27	– – / 0.30
***Bitlet* (float 32)**	9.40 / 67.29	7.36 / 69.97	8.59 / 66.04	4.87 / 60.24
***Bitlet-X* (float 32)**	– – / 92.26	– – / 136.01	– – / 98.20	– – / 86.49

TABLE VI
INFERENCE EFFICIENCY IN THE 16B PRECISION.

Accelerators Baselines	**Inference** efficiency is in GOPs/W			
	lapSRN	DCPDNet	DenseNet -161	FCOS
Eyeriss (16b)	9.92	9.42	9.79	9.71
SCNN (16b)	24.29	20.96	19.49	23.77
Stripes (16b)	25.06	21.34	15.03	19.63
Laconic (16b)	46.04	39.32	31.27	36.64
***Bitlet* (16b)**	168.75	302.71	217.87	221.87
***Bitlet* (float 32)**	44.64	55.82	69.03	50.33
***Bitlet-X* (float 32)**	64.47	84.18	92.08	73.97

Table VI and Table VII show the inference efficiency comparison with fixed-point accelerator baselines. For 16b precision, the improvement over the most recent Laconic is $6.05\times$, $6.97\times$, $7.69\times$, and $3.67\times$. Even for the float 32 precision, both *Bitlet* and *Bitlet-X* behave better than all baselines, which confirms that bit interleaving is more effective than the bit-serial/parallel philosophies. In this way, directly deploying the floating point model on *Bitlet* will also bring satisfactory acceleration.

TABLE VII
INFERENCE EFFICIENCY IN THE 8B PRECISION.

Accelerators Baselines	**Inference** efficiency is in GOPs/W			
	ResNet-50	Mobile -NetV2	YoloV3	Multi -Pose
Eyeriss (8b)	9.79	9.80	9.76	9.85
SCNN (8b)	15.97	15.98	23.88	27.87
Stripes (8b)	14.32	13.03	18.91	20.57
Laconic (8b)	29.60	29.00	35.29	36.58
***Bitlet* (8b)**	236.82	224.13	261.50	202.07
***Bitlet* (float 32)**	62.83	53.92	48.46	52.24
***Bitlet-X* (float 32)**	86.71	80.42	76.07	58.24

TABLE VIII
THE BITLET-X ACCURACY COMPARED WITH EACH PRE-TRAINED MODEL. COMPARATIVE OR HIGHER RESULTS FOR *Bitlet-X* ARE MARKED AS **BOLD**.

Models	**Metric**	**Bitlet (Pretrained Accuracy)**	**Bitlet-X**
ResNet-50	Top-1 (%)	76.13	76.01
MobileNetV2	Top-1 (%)	71.88	71.29
YoloV3	mAP 0.5:0.95	0.336	**0.338**
Multi-Pose	mAP 0.5:0.95	0.59	0.57
lapSRN	PSNR	31.65	31.44
	SSIM	0.90	0.89
DCPDNet	SSIM	0.78	**0.78**
DenseNet-161	Top-1 (%)	77.14	**77.14**
FCOS	mAP 0.5:0.95	0.38	**0.38**
CartoonGAN	See Figure 10		
Transformer	BLEU4	40.83	**40.92**
C3D	Top-1 (%)	97.31	97.27
D3DNet	PSNR	36.05	36.04
	SSIM	0.94	**0.94**

D. Accuracy

The accuracy of *Bitlet* is exactly the same as the pre-trained model because this prototype strictly enforces float-32 precision in computation. The *Bitlet-X* however, aggressively reserves the most significant 8 bits in the mantissa for the sparsity-aware distillation, so it is imperative to evaluate its impact on the final accuracy. In this experiment, we will present data results in combination with the visual comparison for some of the models, because visual comparison is also a common assessment method that directly reflects the user experience. The data results are itemized in Table V-C. The inference on *Bitlet-X* demonstrates around 0.1% and 0.6% Top-1 accuracy fluctuation for the classification models - ResNet-50 and MobileNetV2. For the video understanding model C3D, the Top-1 accuracy drops very slightly from 97.31 to 97.27. Other benchmark models also show competitive accuracy to some extent, in comparison with the pre-trained value. For example, DCPDNet, FCOS, DenseNet-161, and D3DNet apiece show the exactly equivalent accuracy (no accuracy loss), and more promisingly, Transformer on *Bitlet-X* even shows 0.09 higher BLEU4 result; YoloV3 also shows 0.002 higher mAP (0.5:0.95).

We carry out two visual comparisons of the image generation tasks - LapSRN and CartoonGan. Figure 10 compares the cartoon style transfer on *Bitlet* and *Bitlet-X* and the results are nearly equal, demonstrating *Bitlet-X* is a better choice for balancing the trade-off between accuracy and speed in the situation of inference only. A similar result is also shown in Figure 11 for LapSRN.

Discussion: As the well-publicized feature, deep learning models are prone to "over-fitting", because the parameter update during back propagation only aims to learn the distribution of the "training" dataset. The over-fitting problem reflects on the actual parameter value after training. *Bitlet-X* however, changes the distribution of the post-training parameter set, which also counteracts the negative impact of the over-fitting. Therefore, the accuracy even has the chance to increase. This extra bonus also suggests that the end users could feel free to leverage *Bitlet-X* for faster but safe inference for many different deep learning tasks.

Fig. 10. Visual comparison for CartoonGAN. For the original image on *Bitlet* and *Bitlet-X*, respectively, we apply the style transfer. The outcomes demonstrate that *Bitlet-X* was able to infer conclusions more quickly while maintaining the quality of the output.

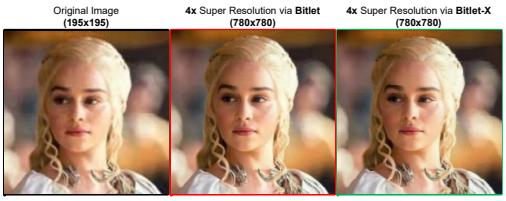


Fig. 11. Visual comparison for lapSRN. Zoom in for a better view. To the original image, 4x super-resolution is applied. The findings for *Bitlet* and *Bitlet-X* are essentially identical, demonstrating their lossless accuracy but faster inference time.

E. Hardware Ablation Study

To explore the impact of each hardware module on the fps, we execute an ablation study, where the target modules contain the preprocessing module that carries out dynamic exponent matching (termed as "**M**") and the *RR-reg* with check window that carries out bit distillation (termed as "**D**"). As shown in Table IX, we have 4 cases in total :

1) If we remove "M" and "D" in tandem (w/o M, w/o D) in *Bitlet*, it is a bare-metal design which is the same as setting N to 1. This case can be configured in both floating- and fixed-point *Bitlet*.
2) If we reserve "M" and remove "D" (w/ M, w/o D), it degenerates from the sparsity-aware to the sparsity-agnostic design, because the distillation phase is disabled and the ineffectual zero bits are still involved in the computation. This case only occurs in the floating-point *Bitlet*.
3) If we remove "M" and reserve "D" (w/o M, w/ D), *Bitlet* can only work at the fixed-point mode since it does not need exponent matching (no exponent in fixed-point values).

TABLE IX
HARDWARE ABLATION STUDY. "BARE-M" INDICATES "BARE-METAL", "**M**" INDICATES DYNAMIC EXPONENT "MATCHING", AND "**D**" INDICATES ESSENTIAL BIT "DISTILLATION". THE RESULTS ARE IN "FPS", AND HIGHER IS BETTER.

Instance	bare-m	***Bitlet-float32***			
Parameters	$N=1$	$N=32$		$N=64$	
Ablation	w/o M w/o D	w/ M w/o D	w/ M w/ D	w/ M w/o D	w/ M w/ D
CartoonGAN	0.012	0.209	0.269	0.244	0.352
Transformer	0.126	2.152	2.879	2.509	3.763
C3D	0.035	0.590	0.771	0.670	0.976
D3DNet	0.003	0.056	0.068	0.065	0.084
Instance	bare-m	***Bitlet-16b***			
Parameters	$N=1$	$N=32$		$N=64$	
Ablation	w/o M w/o D	w/o M w/o D	w/o M w/ D	w/o M w/o D	w/o M w/ D
lapSRN	0.004	0.033	0.040	0.038	0.046
DCPDNet	0.011	0.096	0.184	0.109	0.239
DenseNet-161	0.187	1.567	2.260	1.779	2.812
FCOS	0.036	0.304	0.455	0.345	0.556
Instance	bare-m	***Bitlet-8b***			
Parameters	$N=1$	$N=32$		$N=64$	
Ablation	w/o M w/o D	w/o M w/o D	w/o M w/ D	w/o M w/o D	w/o M w/ D
ResNet-50	0.354	2.970	4.598	3.371	5.793
MobileNetV2	4.730	39.644	60.001	45.001	73.189
YoloV3	0.114	0.959	1.640	1.089	2.066
Multi-Pose	0.030	0.250	0.346	0.284	0.416

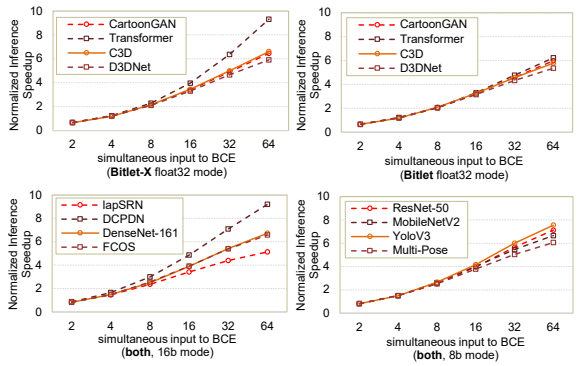


Fig. 12. Sensitivity study for the key design parameter N.

4) If we reserve "M" and "D" in tandem (w/ M, w/ D), it is the standard prototype of the floating-point *Bitlet*.

We emphasize two observations in the ablation results. On the one hand, all *Bitlet* instances perform better than bare-metal. For instance, the speedup of *Bitlet* (float 32, w/ M w/ D) is $29.33\times$ ($N=64$) and $22.41\times$ ($N=32$) for CartoonGAN. Similar to *Bitlet* (8b, w/ M w/ D), it presents $15.47\times$ ($N=64$) and $12.69\times$ ($N=32$) speedup. On the other hand, within *Bitlet* instances, regardless of whether "D" is set or not, bigger N always yields higher frame rates. Taking DCPDNet as the example, the speedup of $N=64$ over $N=32$ is $1.14\times$ and $1.30\times$ for w/o D and w/ D respectively. While in the same N configuration, the w/D over w/o D is $2.19\times$ and $1.92\times$ for $N=64$ and $N=32$ respectively. Therefore, we can conclude that larger N and setting up "D" will both bolster the inference speed, but bit distillation ("D") is the major drive across all the applications and precision studied.

F. Sensitivity of Key Design Parameters

From the ablation study, we find that larger N leads to better fps results. N determines the stride that weights could be interleaved in Figure 4(c). If we set N as large as possible, the number of weights that are simultaneously assimilated by *BCE* is also increased. Additionally, it also provides a larger possibility for distilling the essential bits. As illustrated in Figure 12, we record the inference time for each N scaling from 2 to 64 (power of 2 at each step), the performance increases nearly exponentially for some of the applications, *i.e.,* Transformer, YoloV3 and DCPDNet, and linearly for lapSRN, ResNet-50, and Multi-Pose. Another observation is about the behaviors of the *Bitlet* couple. From the Y-axis in the upper two figures, it is obvious that *Bitlet-X* performs much better than *Bitlet*. This is reasonable because as for the inference, *Bitlet-X* only takes the front 8 bits into account. The essential bits probed by the front 8 *RR-regs* might be fewer, so the inference time is much shorter.

Besides, a larger N will not increase power consumption. Increasing N does not result in an enlargement of the on-chip local buffer because N simply determines how many MACs may be executed simultaneously by single PE. A higher N is preferable if the memory access throughput can commendably match the PE calculation throughput. For this reason, the default configuration was set to $N=64$.

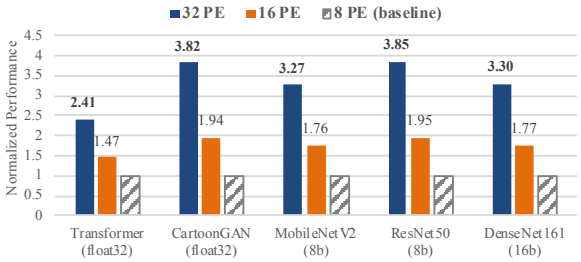


Fig. 13. Sensitivity study to the PE array scale.

G. Scalability

By increasing the number of PE from 8, 16, and 32 in accordance with the performance of the accelerator, we run a scalability analysis, as shown in Figure 13. $Transformer$ is memory intensive, therefore its performance scales $2.41\times$ when using float-32 precision. In addition, other DNNs in the benchmark are computation intensive, therefore more PEs are advantageous for performance improvement. For instance, ResNet50 achieves $3.85\times$ speedup for 32 PEs with 8b precision. Minimized data precision is beneficial to decrease memory accesses, so fixed-precision DNNs possibly exhibit higher performance when PE scales larger.

H. Area and Power Breakdown

In the $28nm$ TSMC technology node, the area of *Bitlet* equipped with 32 PEs in float 32 mode is $1.542mm^2$. Correspondingly, the area is $5.802mm^2$ under the $65nm$ technology node. Table III compares the area of the SOTA accelerators, and *Bitlet* occupies the smallest circuit area. From Table X, we can find the "*Wire Orchestrator & Decoder*" module in *BCE* occupies the largest area (40.1%), because the decoder and some of the wires reorganized are inevitably prolonged to avoid intersection. However, it is not the largest power consumer (only 11.2%), because there isn't a complicated computation circuit in this module. The "preprocessing module" costs

TABLE X
AREA AND POWER BREAKDOWN FOR THE *Bitlet* ACCELERATOR INSTANCES.

Technology Node	**TSMC @28nm**						**TSMC @65nm**					
Instances	**Bitlet (float 32)**		**Bitlet-X (float 32)**	**Bitlet-X (16b)**	**Bitlet-X (8b)**	**Bitlet-X (4b)**	**Bitlet (float 32)**		**Bitlet-X (float 32)**	**Bitlet-X (16b)**	**Bitlet-X (8b)**	**Bitlet-X (4b)**
Metric	**Area** (mm^2)	**Power (mW)**	**Power (mW)**	**Power (mW)**	**Power (mW)**	**Power (mW)**	**Area** (mm^2)	**Power (mW)**	**Power (mW)**	**Power (mW)**	**Power (mW)**	**Power (mW)**
Preprocessing Module	0.553 (35.8%)	356.8 (62.6%)	356.8 (81.4%)	296.598 (68.6%)	296.598 (81.2%)	296.598 (89.28%)	1.916 (33%)	1208.5 (66.1%)	1,208.512 (83.47%)	1000.8 (71.9%)	1000.8 (83.5%)	1000.8 (90.68%)
Wire Orch. & Decoder	0.570 (35.9%)	63.857 (11.2%)	20.651 (4.71%)	40.28 (9.73%)	20.651 (5.54%)	10.3225 (3.11%)	2.327 (40.1%)	164.1 (8.1%)	55.712 (3.85%)	111.4 (8.0%)	55.7 (4.6%)	27.856 (2.52%)
RR-reg & Check Win.	0.131 (9.6%)	27.37 (4.8%)	10.128 (2.31%)	20.28 (4.56%)	10.128 (2.88%)	5.064 (1.52%)	0.7 (12.0%)	112.1 (7.2%)	37.376 (2.58%)	74.8 (5.3%)	37.4 (2.9%)	18.724 (1.7%)
Adder Tree	0.244 (15.8%)	107.424 (18.8%)	35.808 (8.17%)	71.616 (16.6%)	35.808 (9.80%)	17.904 (5.39%)	0.7 (12.0%)	293.3 (16.0%)	97.78 (6.75%)	195.5 (14.1%)	97.8 (9.8%)	48.89 (4.43%)
PostProcessing Module	0.044 (2.9%)	14.88 (2.6%)	14.88 (3.40%)	2.304 (0.53%)	2.304 (0.63%)	2.304 (0.69%)	0.2 (2.9%)	48.5 (2.7%)	48.48 (3.35%)	7.4 (0.5%)	7.4 (0.6%)	7.36 (0.67%)
Total	**1.54**	**570.15**	**438.26**	**432.08**	**365.49**	**332.20**	**5.80**	**1829.60**	**1447.86**	**1390.00**	**1199.10**	**1103.63**

the largest power quota (62.6%) followed by the "adder tree" (18.8%). By comparing the power of preprocessing module, processing floating-point data uses less power than the fixed-point data, but the portion over total power increases from 62.6% to 81.2%. For the *Wire Orchestrator*, lower precision consumes less power as well. Due to the effective clock gating in *Bitlet-X*, the power consumed by the "*Wire Orch.*", "*RR-reg*" and "Adder Tree" is much lower than *Bitlet*. Adding them in total, it shows only $438.26mW$ power consumption for 32 PEs. The point we want to highlight is that *Bitlet*'s power consumption continues to decrease from float-32 to 16b and 8b, demonstrating how highly scalable the architecture is in terms of power.

VI. DISCUSSION

The range of applications for Bitlet. It mainly accelerates the DNNs forward inference. However, since forward propagation is included in DNNs training. Therefore, the Bitlet can also support acceleration in training.

The significance of versatile precision support. The emphasis of Bitlet is to accelerate a variety of DNNs applications generally. However, diverse data types and precision support are needed for different applications (see Table II). As a result, it's critical to offer multi-precision support in a single accelerator, including fixed-point $1b \sim 24b$ and $fp32/16$.

VII. CONCLUSION

In this paper, we propose an effective bit-level technique called "*bit interleaving*" and the related accelerator design namely "*Bitlet*" for general-purpose deep learning acceleration. It leverages the sparsity parallelism in the parameters and implements "dynamic exponent matching" and "essential bit distillation" to avoid the pointless computations that could possibly slow down the inference performance. *Bitlet* is flexible by supporting both the fixed-point ($1 \sim 24$b) and floating-point (fp32/fp16) precision. Users could investigate the optimum accuracy/speedup/power tradeoff by testing their models at any precision, saving time and allowing for quicker deployment. We believe that the proposed techniques can provide new chances for researchers to explore novel applications in deep learning and even algorithms beyond AI. We also hope it stimulates fresh thoughts regarding the design of deep learning accelerators, by applying the same concept in conjunction with some optimization techniques like pruning, and on other hardware platforms (*i.e.,* GPGPUs) in the future.

REFERENCES

[1] "Coco dataset." [Online]. Available: https://cocodataset.org/\#download
[2] "Flickr image dataset." [Online]. Available: https://www.kaggle.com/hsankesara/flickr-image-dataset
[3] "Imagenet large scale visual recognition challenge." [Online]. Available: http://www.image-net.org/challenges/LSVRC
[4] "Set 14." [Online]. Available: https://github.com/jbhuang0604/SelfExSR
[5] "Ucf101 – action recognition data set." [Online]. Available: https://www.crcv.ucf.edu/research/data-sets/ucf101
[6] "Wmt dataset." [Online]. Available: http://data.statmt.org
[7] J. Albericio, A. Delmás, P. Judd, S. Sharify, G. O'Leary, R. Genov, and A. Moshovos, "Bit-pragmatic deep neural network computing," in *MICRO 2017*.
[8] Y. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.
[9] Y. Chen, Y. Lai, and Y. Liu, "Cartoongan: Generative adversarial networks for photo cartoonization," in *CVPR 2018*.
[10] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "Dadiannao: A machine-learning supercomputer," in *MICRO 2014*.
[11] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," *ISCA 2016*.
[12] M. S. Committee *et al.*, "754-2019-ieee standard for floating-point arithmetic," 2019.
[13] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: Efficient inference engine on compressed deep neural network," in *ISCA 2016*.
[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR 2016*.
[15] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *CVPR 2017*.
[16] S. Hurkat and J. F. Martínez, "Vip: A versatile inference processor," in *HPCA 2019*.
[17] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos, "Stripes: Bit-serial deep neural network computing," in *MICRO 2016*.
[18] M. Jung, C. Weis, and N. Wehn, "Dramsys: A flexible dram subsystem design space exploration framework," *IPSJ Transactions on System LSI Design Methodology*, vol. 8, pp. 63–74, 2015.
[19] M. Kocabas, S. Karagoz, and E. Akbas, "Multiposenet: Fast multi-person pose estimation using pose residual network," in *ECCV 2018*.
[20] W. Lai, J. Huang, N. Ahuja, and M. Yang, "Deep laplacian pyramid networks for fast and accurate super-resolution," in *CVPR 2017*.
[21] A. D. Lascorz, P. Judd, D. M. Stuart, Z. Poulos, M. Mahmoud, S. Sharify, M. Nikolic, K. Siu, and A. Moshovos, "Bit-tactical: A software/hardware approach to exploiting value and bit sparsity in neural networks," in *ASPLOS 2019*.
[22] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H. Yoo, "Unpu: An energy-efficient deep neural network accelerator with fully variable weight bit precision," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 1, pp. 173–185, 2019.
[23] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," arXiv/1608.08710, 2016.
[24] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *ICCV 2017*.

[25] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," *arXiv preprint arXiv:1611.06440*, 2016.
[26] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "Scnn: An accelerator for compressed-sparse convolutional neural networks," in *ISCA 2017*.
[27] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement." *arXiv: Computer Vision and Pattern Recognition*, 2018.
[28] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *CVPR 2018*.
[29] S. Sharify, A. D. Lascorz, M. Mahmoud, M. Nikolic, K. Siu, D. M. Stuart, Z. Poulos, and A. Moshovos, "Laconic deep learning inference acceleration," in *ISCA 2019*.
[30] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, V. Chandra, and H. Esmaeilzadeh, "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network," in *ISCA 2018*.
[31] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus, "Indoor segmentation and support inference from rgbd images," in *ECCV 2012*.
[32] Z. Tian, C. Shen, H. Chen, and T. He, "Fcos: Fully convolutional one-stage object detection," in *ICCV 2019*.
[33] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3d convolutional networks," in *ICCV 2015*.
[34] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.
[35] T. Xue, B. Chen, J. Wu, D. Wei, and W. T. Freeman, "Video enhancement with task-oriented flow," *International Journal of Computer Vision*, vol. 127, no. 8, pp. 1106–1125, 2019.
[36] L. Yang, Z. He, and D. Fan, "Harmonious coexistence of structured weight pruning and ternarization for deep neural networks," in *AAAI 2020*.
[37] X. Ying, L. Wang, Y. Wang, W. Sheng, W. An, and Y. Guo, "Deformable 3d convolution for video super-resolution," *IEEE Signal Processing Letters*, vol. 27, pp. 1500–1504, 2020.
[38] H. Zhang and V. M. Patel, "Densely connected pyramid dehazing network," in *CVPR 2018*.
[39] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen, "Cambricon-x: An accelerator for sparse neural networks," in *MICRO 2016*.
[40] X. Zhou, Z. Du, Q. Guo, S. Liu, C. Liu, C. Wang, X. Zhou, L. Li, T. Chen, and Y. Chen, "Cambricon-s: Addressing irregularity in sparse neural networks through a cooperative software/hardware approach," in *MICRO 2018*.

Liang Chang (M'19) received the Ph.D. and M.S. degrees from Beihang University in 2019 and 2014, respectively. He received a double B.S. degree from the Chengdu University of Information and Technology and the University of Electronic Science and Technology of China 2011. He was an engineer and senior engineer in China Glorun Technology (Beijing, China) and AMD China (Beijing, China) during 2012-2015. Since 2020, he has been an Associate Professor at the School of Information and Communication Engineering, University of Electronic Science and Technology of China. From 2022, he was a visiting scholar at HKUST. He has co-authored over 50 scientific papers, including IEEE International Solid-State Circuits Conference (2021, 2023, 2024), MICRO (2021), IEEE TCAS-I (2020-2024), IEEE TVLSI (2019, 2024), IEEE TC (2019), IEEE TCAD, etc. His research interests include computing in emerging nonvolatile memory, advanced memory-centric computer architecture, and AI processors for intelligent detection. He is the Regular Reviewer of the IEEE JSSC, IEEE TCAS-I/II, IEEE TBioCAS, IEEE TCAD, IEEE TC, IEEE TVLSI, etc.

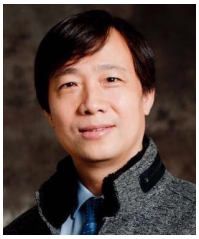
Hang Lu currently an Associate Professor and Master Tutor at the State Key Laboratory of Computer Architecture, Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS). He is also a research scientist with the Shanghai Innovation Center for Processor Technologies. He is a member of the Youth Innovation Promotion Association of CAS, and the New Best Star of ICT. His research interests include power-efficienct computing platforms, AI chip design, deep learning algorithm optimization, etc.

Chenglong Li Received B.Eng degree from UESTC, Chengdu, China, in 2020. He is currently working toward the M.Eng degree under the guidance of Prof Liang Chang and Shuisheng Lin at UESTC, Chengdu, China. His research interests are high-performance circuit design and computing-in-memory architecture.

Xin Zhao received the B.E. degree in Electronic Engineering from China University of Mining and Technology, Xuzhou, China, in 2021. He is currently pursuing the M.S. degree under the guidance of Prof Liang Chang from the Department of Internet of things Engineering, School of information and Communication Engineering, University of Electronic Science and Technology of China. His research interest contains computing-in-memory architecture and super-resolution hardware architecture.

Zhicheng Hu received the B.E. degree in microelectronics from University of Electronic Science and Technology of China, Chengdu, China, in 2022, where he is pursuing the M.S. degree under the guidance of Prof Liang Chang from the Department of Internet of Things Engineering, School of Information and Communication Engineering. His current research interests include energy-efficient deep-neural-network architectures/accelerators especially for image processing region and efficient deep learning based algorithm for hardware processing.

Jun Zhou (Senior Member, IEEE) received the dual B.S. degree in communication engineering and microelectronics from the University of Electronic Science and Technology of China (UESTC) in 2004 and the Ph.D. degree in microelectronics system design from Newcastle University, U.K., in 2008. He joined IMEC Netherlands in 2008 as a Research Scientist and worked on the energy-efficient processor design for intelligent sensing in collaboration with companies, such as Philips and NXP. In 2011, he joined the Institute of Microelectronics (IME), Agency for Science, Technology, and Research (ASTAR), Singapore, where he led projects and supervises Ph.D. students on energyefficient processor design for intelligent sensing. In 2017, he joined UESTC as a Professor and is currently leading the Research Group of Smart ICs and Systems for IoT Applications. His major research interest is processor and algorithm codesign for intelligent sensing. He has published more than 80 papers in prestigious conferences and journals, including ISSCC, JSSC, DAC and CICC.



Xiaowei Li (SM'04) received the B.Eng. and M.Eng. degrees in computer science from the Hefei University of Technology, Hefei, China, in 1985 and 1988, respectively, and the Ph.D. degree in computer science from the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing, China, in 1991. He was an Associate Professor with the Department of Computer Science and Technology, Peking University, Beijing, from 1991 to 2000. In 2000, he joined ICT, CAS, as a Professor, where he is currently the Deputy Director of the State Key Laboratory of Computer Architecture. He has coauthored over 280 papers in journals and international conferences, and he holds 60 patents and 30 software copyrights. His current research interests include VLSI testing, design for testability, design verification, dependable computing, and wireless sensor networks. He services as an Associate Editor for the Journal of Computer Science and Technology, the Journal of Low Power Electronics, the Journal of Electronic Testing: Theory and Applications, and the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS.