electronics

Article

TORRES: A Resource-Efficient Inference Processor for Binary Convolutional Neural Networks Based on Locality-Aware Operation Skipping

Su-Jung Lee †, Gil-Ho Kwak † and Tae-Hwan Kim *,†

School of Electronics and Information Engineering, Korea Aerospace University, 76, Hanggongdaehak-ro, Deogyang-gu, Goyang-si 10540, Korea
* Correspondence: taehwan.kim@kau.kr
† These authors contributed equally to this work.

Abstract: A binary convolutional neural network (BCNN) is a neural network promising to realize analysis of visual imagery in low-cost resource-limited devices. This study presents an efficient inference processor for BCNNs, named TORRES. TORRES performs inference efficiently, skipping operations based on the spatial locality inherent in feature maps. The training process is regularized with the objective of skipping more operations. The microarchitecture is designed to skip operations and generate addresses efficiently with low resource usage. A prototype inference system based on TORRES has been implemented in a 28 nm field-programmable gate array, and its functionality has been verified for practical inference tasks. Implemented with 2.31 K LUTs, TORRES achieves the inference speed of 291.2 GOP/s, exhibiting the resource efficiency of 126.06 MOP/s/LUT. The resource efficiency of TORRES is 1.45 times higher than that of the state-of-the-art work.

Keywords: binary convolutional neural networks; processor; inference; resource efficiency; field-programmable gate array

check for updates

Citation: Lee, S.-J.; Kwak, G.-H.; Kim, T.-H. TORRES: A Resource-Efficient Inference Processor for Binary Convolutional Neural Networks Based on Locality-Aware Operation Skipping. *Electronics* **2022**, *11*, 3534. https://doi.org/10.3390/electronics11213534

Academic Editor: Xiang Chen

Received: 22 September 2022
Accepted: 14 October 2022
Published: 29 October 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

1. Introduction

A binary convolutional neural network (BCNN) is a class of convolutional neural networks (CNNs) in which every element of the weights and features is quantized to a single bit [1]. Such binarization may reduce the memory bandwidth required to achieve high inference speed. In addition, it may facilitate strength reduction in performing convolutions as the computation can be carried out with single-bit elements [2]. Interestingly, the inference performance (e.g., the accuracy in the classification tasks) is close to that achievable by the conventional full-precision CNN, particularly for small- or medium-scale tasks, if trained based on the straight-through backward propagation of the gradients for non-differentiable binarization functions [1,2]. With these merits, BCNNs are widely employed for various applications to achieve efficient inference in low-cost devices [3,4].

There are several BCNN inference processors presented in the previous studies. XNOR Neural Engine is an efficient compute engine for BCNN inference, developed to be integrated into a tiny microcontroller unit (MCU) [5]. BinarEye is based on a flexible memory-like binary neuron array with every weight on a chip [6]. FBNA is based on a modified BCNN whose inference can be performed in a fully binary domain [7]. The BCNN inference processor presented in [8] skips the redundant operations involved in pooling windows and padded zeros to achieve high resource efficiency. IOTA is an inference system developed by integrating dual processing cores, which are controlled to work in a cooperative manner [9]. O3BNN-R is based on a novel out-of-order architecture in which each processing core works with variable latency skipping operations [10]. MajorityNet is an efficient BCNN modified by employing majority operations to achieve low resource usage [11]. CUTIE is a ternary CNN inference processor designed based on an unrolled architecture to achieve very high

speed [12]. The BCNN inference processor presented in [13] achieves high inference speed effectively based on the novel channel-aware operations with the group-pruning technique. The global average pooling, which can replace several fully-connected blocks so as to reduce the number of the weight elements, is realized by a majority circuit in the BCNN inference processor presented in [14]. (BinarEye, FBNA, IOTA, O3BNN-R, and CUTIE correspond to the names of the BCNN inference processors developed from the previous studies stated here.)

There are some opportunites to skip operations in performing the BCNN inference process. Such opportunites are provided on the basis of the fact that each feature element is represented by a single bit. Some of the previous BCNN inference processors [8–10] stated above exploits the opportunites to skip operations based on the threshold-based and pooling-based techniques, which are going to be described in more detail in the next section. We envision that further opportunities to skip operations can be found considering the spatial locality inherent in feature maps, and this has motivated this study.

This study presents the design and implementation of an efficien**T** BCNN inference pr**O**cesso**R** based on locality-awa**RE** operation **S**kipping, named TORRES.

- TORRES efficiently skips some operations within the pooling windows to achieve high inference speed by exploiting the spatial locality inherent in feature maps. Furthermore, the training process is regularized by modifying the loss function so that more operations can be skipped.
- The microarchitecture is designed to skip operations without considerable resource overhead. In addition, address generation is carried out efficiently by carefully ordering the elements in the memories.
- A prototype inference system has been implemented based on TORRES in a 28 nm field-programmable gate array (FPGA), under which the functionality has been verified elaborately for practical inference tasks. The resource efficiency of TORRES is as high as 126.06 MOP/s/LUT at the inference speed of 291.2 GOP/s with the resource usage of 2.31 K LUTs. The inference accuracy is 87.47% for the CIFAR10 classification task.

The rest of the paper is organized as follows. Section 2 briefly describes the conventional BCNN inference process. Section 3 presents TORRES in detail by describing its processing flow and microarchitecture. Section 4 presents and evaluates the implementation results. Finally, Section 5 draws the conclusion.

2. Background

In this study, XNOR-Net is employed for the BCNN models because it is feasible to realize a fully binarized inference process without involving any real operations [2]. (In fact, there are a few of BCNN models such as those presented in [15,16], achieving superior inference performance than XNOR-Net. However, they are not suitable to implement an efficient inference processor because their inference process cannot be simply decomposed into the blocks for which each element of the input and output data is binarized.) The overall inference process can be divided into several blocks that share such a computing procedure that is illustrated in Figure 1. For each channel of the output feature map, the input feature map is first convolved with the weights in the corresponding channel; the resulting feature map is binarized by comparing it to a threshold, and the output feature map is finally computed through optional pooling. Here, the dot products for the convolution can be calculated based on bitwise XNOR operations followed by population counting, which is called an XPOP operation [2], symbolized by $\odot$ in Figure 1; the threshold can be precomputed with the real weights and batch normalization parameters [5,10,17]; max pooling is usually employed to decimate the feature map [18]. The computing procedures for the fully connected blocks can also be described by that shown in Figure 1 with the unity spatial dimensions.

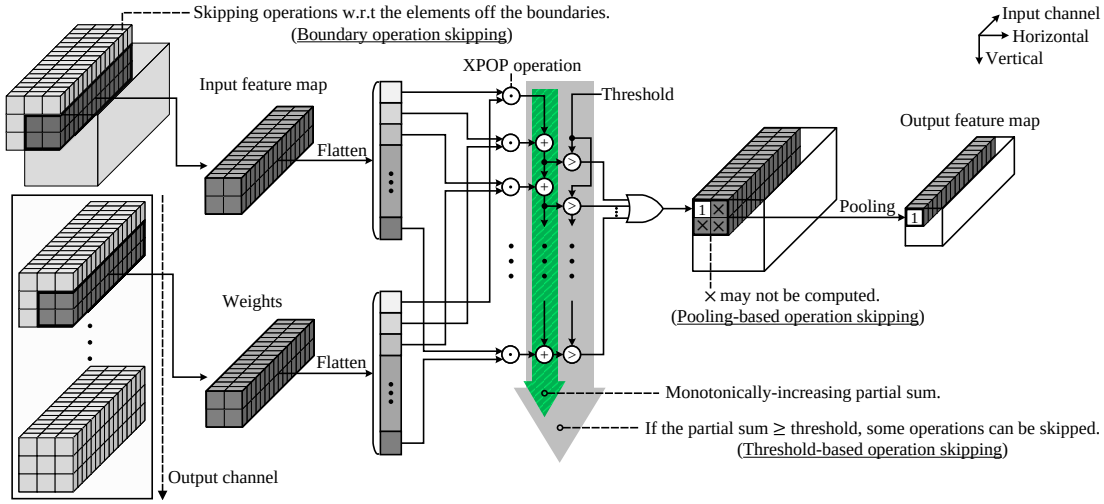


Figure 1. Conventional procedure for computing each block composing the BCNN inference process, where the input and output feature maps and weights are multi-dimensional tensors in which each element is represented by a single bit.

Several previous schemes can be employed to perform the procedure efficiently by skipping operations.

- Threshold-based operation skipping [8,10]: The partial sum monotonically increases as calculated by accumulating the non-negative results of the XPOP operations. Therefore, the resulting bit can be determined immediately after the partial sum is greater than a threshold, and the remaining operations can be skipped.
- Pooling-based operation skipping [8,10]: Max pooling can be performed based on the logical OR operations for the binarized elements within a window [19]; hence, the bit resulting from the window can be determined immediately after any element within the window is determined to be 1, and the operations involved in computing the other elements can be skipped.
- Boundary operation skipping [8]: The operations with the zeros padded off the boundaries of the input feature map to maintain the size across the convolution can be skipped by trimming the receptive fields.

3. Proposed Processor: TORRES

3.1. Processing Flow

TORRES is designed to focus on efficient processing of each block. Algorithm 1 delineates the processing flow, where the subscripts index the elements in a tensor, range$(0, a)$ is the tuple $(0, 1, \cdots a-1)$, and $\mathbf{C} \star \mathbf{D}$ is the Cartesian product of the two tuples $\mathbf{C}$ and $\mathbf{D}$, that is, $((a, b)|a \in \mathbf{C}, b \in \mathbf{D})$. In notating the multi-dimensional tensors, the dimensions are arranged in the order of the horizontal, vertical, input, and output channels. The weight is a binary tensor, whose shape in each input channel is a square of the size $k \times k$. The processing flow is output-stationary [20], where the results of the XPOP operations with the α-bit vectors are accumulated to the partial sum λ through iterations, as described by Line 14. $I_{\gamma+u,\delta+v,\alpha t \cdots \alpha t+\alpha-1}$ and $F_{u+\lfloor \frac{k}{2} \rfloor, v+\lfloor \frac{k}{2} \rfloor, \alpha t \cdots \alpha t+\alpha-1, d}$ are the vectors of α elements located through the input channels in $\mathbf{I}$ and $\mathbf{F}$, respectively, where $a \cdots b$ denotes the vector composed of the values from a to b, inclusively, $a \leq b$, and $\lfloor \cdot \rfloor$ represents the floor function. As for the dimension sizes, w and h are considered integer multiples of s and c and c' are considered integer multiples of α, without loss of generality. The threshold-based and pooling-based operation-skipping schemes [8,10] are implemented as described by Lines 15–17, where the loop is terminated immediately after the resulting bit is determined.

The boundary operation-skipping scheme [8] is implemented by trimming the receptive field and biasing λ initially, as described by Lines 11–12, where

$$\lambda_{\text{bias}} = -\left(\min\left(h-\delta-1, \left\lfloor\frac{k}{2}\right\rfloor\right) + \min\left(\delta, \left\lfloor\frac{k}{2}\right\rfloor\right) + 1\right) \cdot \left(\min\left(w-\gamma-1, \left\lfloor\frac{k}{2}\right\rfloor\right) + \min\left(\gamma, \left\lfloor\frac{k}{2}\right\rfloor\right) + 1\right) \cdot c, \tag{1}$$

$$\mathbf{Q} = \text{range}\left(-\min\left(\gamma, \left\lfloor\frac{k}{2}\right\rfloor\right), \min\left(w-\gamma-1, \left\lfloor\frac{k}{2}\right\rfloor\right) + 1\right) \star \text{range}\left(-\min\left(\delta, \left\lfloor\frac{k}{2}\right\rfloor\right), \min\left(h-\delta-1, \left\lfloor\frac{k}{2}\right\rfloor\right) + 1\right). \tag{2}$$

Here, $\min(\cdot)$ represents the minimum function. The details how the expressions have been formulated as above can be found in [8].

The processing flow is enhanced by another novel operation-skipping scheme. In the BCNN inference process, features are represented in a two-dimensional vector in each channel, so called the feature map. Each feature map can be considered a kind of images, in which adjacent elements are correlated to each other to present some structures together. Such correlations between the data located closely are known as the *spatial locality* in a feature map. Given with an element that has been determined to be a certain bit, we can estimate the relative order of the likelihoods of its neighboring elements to be this bit according to the distances from them to this element based on the spatial locality. The proposed scheme takes the spatial locality into account for skipping operations using two techniques: *compute ordering* and *zero prediction*.

The compute order in a pooling window is determined to find any element of 1 as early as possible with the aim of skipping more operations by the pooling-based scheme. The spatial locality considered by the adjacent window on the immediate left of the current window is exploited for this purpose. Figure 2 illustrates the concept of this technique. If the bit resulting from the adjacent window is 1, the elements located close to the element of 1 in the adjacent window are first computed. This is because they are more likely to be 1's than the others, based on the spatial locality. If the bit resulting from the adjacent window is 0, the elements located far from the adjacent window are computed first. This is because they are relatively less likely to be 0's than the others (i.e., more likely to be 1's), based on the spatial locality. Figure 3 illustrates the compute order for the pooling window of size 2×2. Line 6 in Algorithm 1 describes the compute ordering based on the spatial locality considered by the bit resulting from the adjacent window and its location therein, which are represented by $O_{x-1,y,d}$ and ρ_d, respectively.

With the compute ordering technique applied, if some elements within a window have already been determined to be 0's, the other remaining elements are likely to be 0's; so that the operations involved in computing them can be skipped effectively by predicting the resulting bit to 0. Such prediction may become more accurate if the bit resulting from the adjacent window is also 0. Figure 4 illustrates the concept of this technique. Here, the bit resulting from the adjacent window is 0 and τ elements in the current window have already been determined to be 0's, where the window size is $s \times s$ and $\tau \in \{1, \cdots, s^2\}$. Since the $s^2 - \tau$ elements remaining in the current window are located between the elements of 0's belonging to the two adjoining windows, they are likely to be 0's based on the spatial locality, as illustrated in the figure. Therefore, the resulting bit can be predicted to be 0, and the remaining operations can be skipped. Lines 19–21 in Algorithm 1 describes the zero prediction, where the number of the elements that have been computed within the window, denoted by θ, is compared with τ.

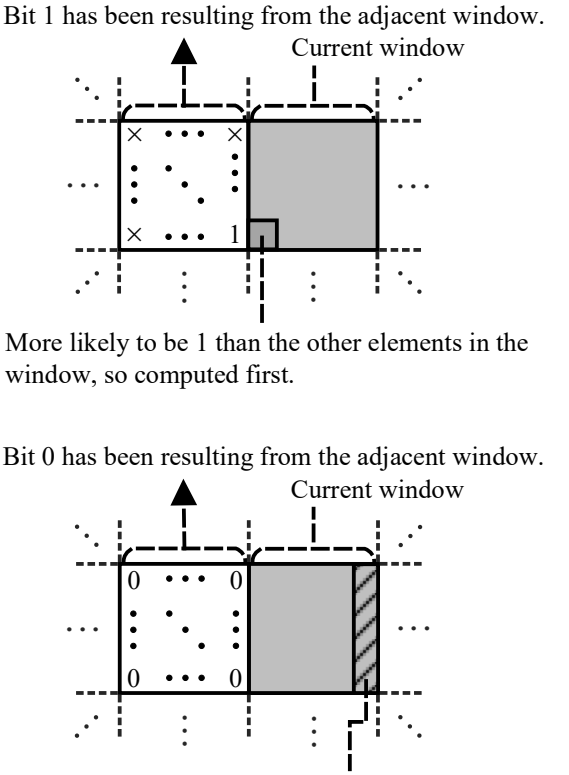


Figure 2. Compute ordering technique.

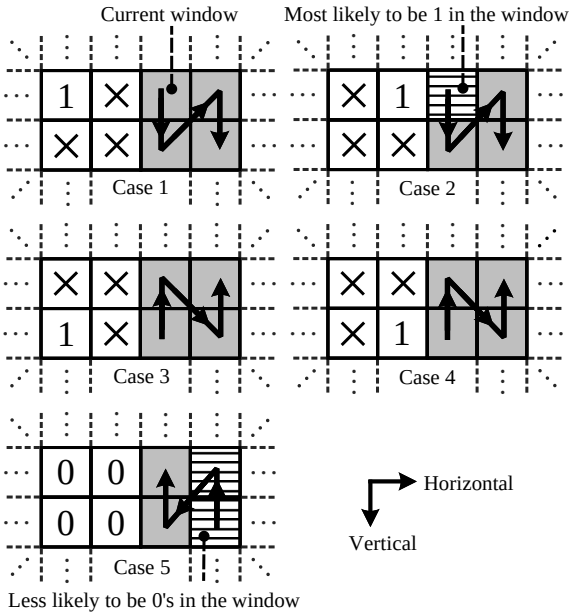


Figure 3. Compute order for each case in the pooling window of size 2 × 2.

Algorithm 1 Block processing flow in TORRES, where $\mathcal{B} \triangleq \{0,1\}$, and $\mathcal{N}$ and $\mathcal{Z}$ are the natural and integer number sets, respectively

Input: input feature map $\mathbf{I} \in \mathcal{B}^{w \times h \times c}$, weight $\mathbf{F} \in \mathcal{B}^{k \times k \times c \times c'}$, pooling window size $s \in \mathcal{N}$, threshold $\mathbf{T} \in \mathcal{Z}^{c'}$, operation-skipping parameters $\eta \in \mathcal{N}$ and $\tau \in \{1, \cdots, s^2\}$

Output: output feature map $\mathbf{O} \in \mathcal{B}^{\frac{w}{s} \times \frac{h}{s} \times c'}$

1: **for** $(x, y) \in \text{range}(0, \frac{w}{s}) \star \text{range}(0, \frac{h}{s})$ **do** ▷ Loop 0: $\mathbf{O}$ ($\rightarrow\downarrow$)
2: **for** $d \in \text{range}(0, c')$ **do** ▷ Loop 1: $\mathbf{O}$ ($\nearrow$)
3: $O_{x,y,d} \leftarrow 0$
4: $\mathbf{S} \leftarrow \text{range}(0, s) \star \text{range}(0, s)$
5: **if** $x > 0$ **then**
6: Reorder $\mathbf{S}$ based on the spatial locality with $O_{x-1,y,d}$ and ρ_d (e.g., Figure 3 for the 2×2 pooling window)
7: $\theta \leftarrow 0$ ▷ θ: no. elements computed within the pooling window
8: **for** $(m, n) \in \mathbf{S}$ **do** ▷ Loop 2: pooling window
9: $\rho_d \leftarrow (m, n)$ ▷ ρ_d: location within the window in the d-th channel
10: $(\gamma, \delta) \leftarrow (sx + m, sy + n)$
11: $\lambda \leftarrow \lambda_{\text{bias}}$ ▷ Bias λ initially [8]
12: **for** $(u, v) \in \mathbf{Q}$ **do** ▷ Loop 3: receptive field in $\mathbf{I}(\rightarrow\downarrow)$; possibly trim the ranges of (u, v) [8]
13: **for** $t \in \text{range}(0, \frac{c}{\alpha})$ **do** ▷ Loop 4: receptive field in $\mathbf{I}(\nearrow)$
14: $\lambda \leftarrow \lambda + 2I_{\gamma+u,\delta+v,\alpha t \cdots \alpha t+\alpha-1} \odot F_{u+\lfloor \frac{k}{2} \rfloor, v+\lfloor \frac{k}{2} \rfloor, \alpha t \cdots \alpha t+\alpha-1, d}$ ▷ XPOP op. for the subvectors
15: **if** $\lambda > T_d$ **then**
16: $O_{x,y,d} \leftarrow 1$
17: Terminate Loop 2 and go to Line 3
18: $\theta \leftarrow \theta + 1$
19: **if** (η is 1 or $x\%\eta < \eta - 1$), $x > 0$, $\theta \geq \tau$, and $O_{x-1,y,d}$ is 0 **then**
20: $O_{x,y,d} \leftarrow 0$
21: Terminate Loop 2 and go to Line 3

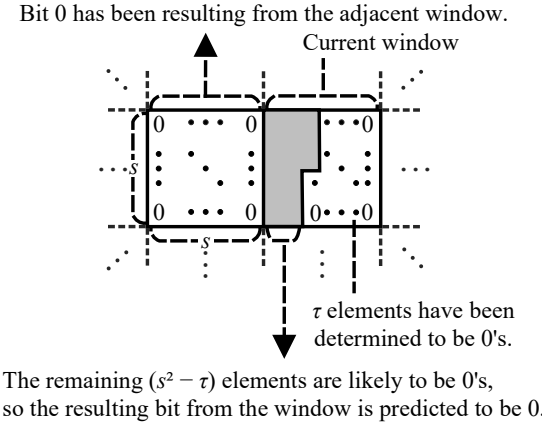


Figure 4. Zero prediction technique.

It is worth noting that the prediction relies on the bit resulting from the adjacent window, which could also have been a predicted one. Thus, a misprediction could propagate through windows. To limit the propagation of misprediction, the prediction is not

performed for the last window for each series of η windows with $\eta > 1$, as described by Line 19 in Algorithm 1, where % represents the modulo operation.

Noteworthy remarks are followed.

- To find the spatial locality, the proposed scheme is designed to consider only one adjacent window located to the immediate left of the current window. According to the processing flow in Algorithm 1, this window corresponds to the previous window that has been processed just before the current window for the feature map in a channel. If more windows neighboring the current window were considered, the locality could be found more *meticulously*. However, it may incur a substantial complexity to find the spatial locality by considering more windows, which may lead to a prohibitive resource overhead for the implementation.
- In the compute ordering technique, the ordering complexity increases with the window size. Therefore, the resource overhead may be considerable for the implementation of the technique to support larger windows. However, in practical BCNN models, such as those presented in [3,14,18,19], the pooling window size is not usually so large since there may be a drastic loss of features through a large pooling window. The ordering for such a small window as illustrated in Figure 3 involves low complexity with only a few cases.
- There are two parameters of the zero-prediction technique: τ and η. Configured for each block-processing, these parameters can be used to control the effects on the inference results as well as the number of operations to be skipped. More specifically, configuring them to smaller values results in a greater deviation of the inference results from those obtained without applying the technique of skipping more operations.

The BCNN models can be trained using a regularization technique to skip more operations. If more elements were determined to be 0's in the output feature maps, we would have fewer opportunities for skipping operations. This is because the threshold-based and pooling-based schemes can skip operations with respect to the elements to be 1's. Motivated by this, the loss to be minimized by the training process is modified by adding a regularization term to penalize such situations that there are many 0's in the output feature maps, as expressed below.

$$L + \omega \cdot \sum_{i \in \{0,\cdots,N-1\}} \sum_{(x,y,z) \in \mathbf{U}^{(i)}} \left(1 - O^{(i)}_{x,y,z}\right), \tag{3}$$

where L is the conventional loss (e.g., cross-entropy loss in the classification tasks), $O^{(i)}_{x,y,z}$ is the bit located at (x, y, z) in the output feature map for the i-th block, N is the number of the blocks, $\mathbf{U}^{(i)}$ is the set of valid locations in the output feature map for the i-th block, and ω is the scaling factor for the regularization term.

3.2. Microarchitecture

The microarchitecture is designed to process each block efficiently according to the flow in Algorithm 1. Figure 5 shows the five stages of the processing pipeline, which can execute each iteration per cycle in the innermost loop in Algorithm 1 for $\alpha = 128$. Stage 1 generates the addresses for accessing the memories storing the thresholds, features, and weights with the counter values corresponding to the loop variables in Algorithm 1. Stage 2 reads the data from the memories. Stages 3 and 4 carry out the XPOP operation with the vectors of 128 elements read from the memories. The summation of the bitwise XNOR operation is first compressed to a carry-save form in Stage 3 and then accumulated to the register storing the partial sum in Stage 4. The resulting bit is determined based on the partial sum with threshold. The shift-register in Stage 4 buffers the resulting bit and its location within the pooling window in the channel direction. Stage 5 writes the resulting bits stored in the shift register to the memory.

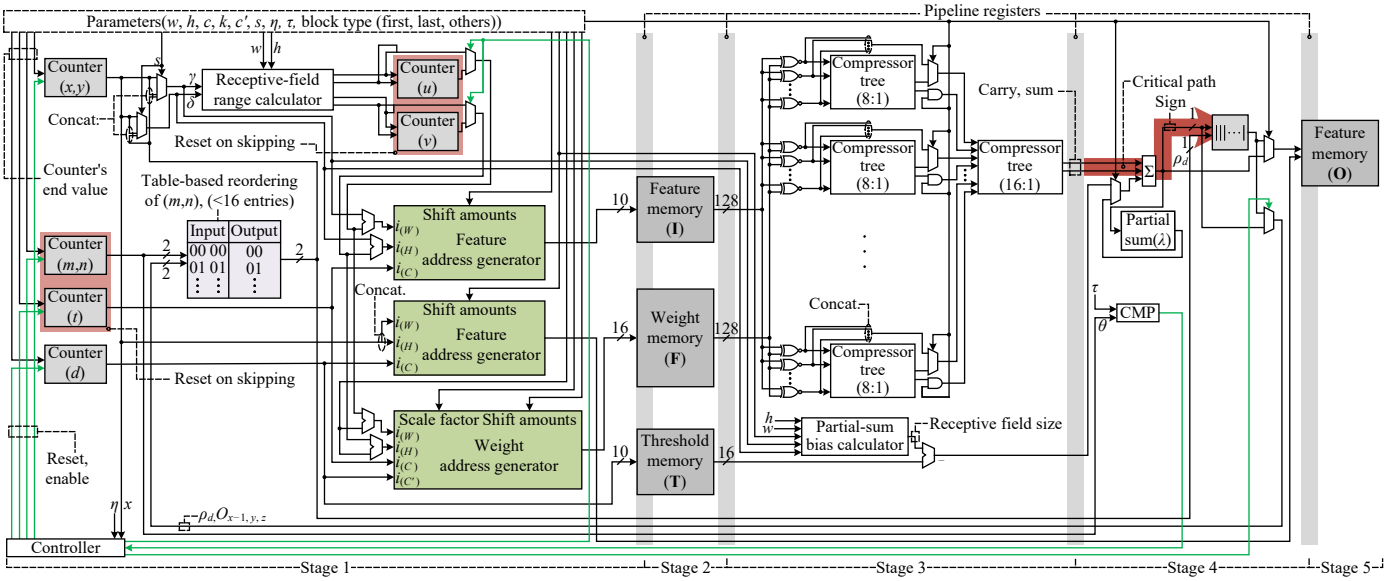


Figure 5. Processing pipeline for Algorithm 1, where CMP stands for a comparator and the variables correspond to those in Algorithm 1. The control signals have been colored green.

The proposed operation-skipping scheme is implemented efficiently without considerable resource overhead. The supported pooling window size is determined to 2×2 by considering the structures of the practical models, as discussed in the previous subsection. The compute ordering technique is realized by reordering the counter values (m, n) based on the locality before using them for address generation, where the reordering is carried out by looking up the small table. The operation skipping by the zero-prediction technique is realized by controlling the counters with the related conditions, which include that induced by the comparison of τ and θ. The locality is considered by the bit resulting from the adjacent window and its location therein, which are stored in and retrieved from the shift register in Stage 4. The location is represented by a single bit because it is only necessary to distinguish whether the resulting bit is located at the upper or lower part in determining the compute order for the pooling window of size 2×2, as illustrated in Figure 3. It should be noted that the additional components employed to implement the proposed scheme do not lie in the critical path highlighted in Figure 5.

The addresses for accessing the memories are efficiently generated from the indices of the elements in the tensors. Address generation depends on how the elements are ordered in the *contiguous* regions of the memories. The dimensions of the weights and features can be considered powers of two without loss of generality except the spatial dimensions of the weights; they are usually odd for realizing symmetric shapes [3,18,19]. Hence, the address generation for the weight memory cannot be carried out by simply concatenating the indices, while the address generation for the feature memories can be carried out so. For TORRES, the ordering type for the weight elements are determined carefully so that the dimensions of non-power-of-two sizes (i.e., spatial dimensions) are in the major. Table 1 shows the ordering type for the weight elements in TORRES, along with the two of traditional ordering types. Here, the channel dimensions, indexed by $i_{(C)}$ and $i_{(C')}$, are powers of two while the spatial dimensions, indexed by $i_{(W)}$ and $i_{(H)}$, are not. The input and output channel dimensions are c and c', respectively, and the spatial dimensions are k, according as Algorithm 1 describes. The ordering type in TORRES facilitates address generation by concatenating $i_{(C)}$ and $i_{(C')}$ with that calculated by $i_{(W)}$ and $i_{(H)}$, according to the layout shown in Table 1. This is more efficient than the traditional ones in terms of the computational complexity. Figure 6 shows the internal structures of the address generators. They contain a few of the barrel shifters and logical OR gates to concatenate the indices in dimensions of variable sizes. The address generator for the weight memory contains one multiply-accumulate (MAC) unit to carry out the calculation with the indices

in the dimensions of non-power-of-two sizes. The shift amounts and scaling factor for the address generators are given by taking account of the shapes of the tensors for each block processing.

Table 1. Weight address generation.

Element ordering [a]	$i_{(W)} \to i_{(H)} \to i_{(C)} \to i_{(C')}$ (Traditional [21–23])	$i_{(C)} \to i_{(W)} \to i_{(H)} \to i_{(C')}$ (Traditional [8,9,24])	$i_{(C)} \to i_{(C')} \to i_{(W)} \to i_{(H)}$ (TORRES)
Address layout [b]	$k^2c \cdot i_{(C')} + k^2 \cdot i_{(C)} + k \cdot i_{(H)} + i_{(W)}$	$k^2 \cdot i_{(C')} + k \cdot i_{(H)} + i_{(W)}$ | $i_{(C)}$	$k \cdot i_{(H)} + i_{(W)}$ | $i_{(C')}$ | $i_{(C)}$
Computational complexity [c]	Three MACs	Two MACs	One MAC

[a] The indices in the spatial dimension are denoted by $i_{(W)}$ (horizontal) and $i_{(H)}$ (vertical) and those in the input and output channel dimensions are denoted by $i_{(C)}$ and $i_{(C')}$. The right ones are major to the left ones in the ordering. [b] The sizes of the fields are determined by the sizes of the dimensions. [c] k and c are constant for each block processing, so are k^2c and k^2; the complexity has been thus estimated given with every of coefficients precomputed.

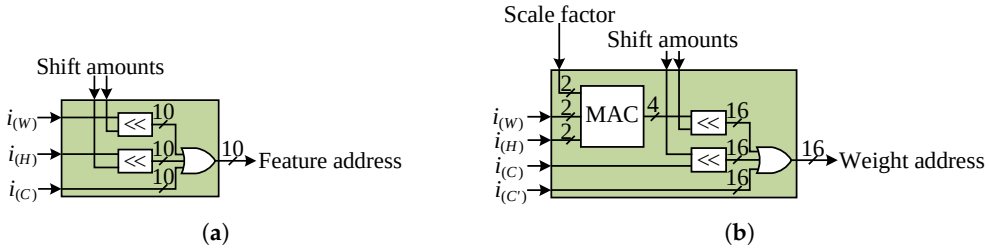


Figure 6. Address generators for the (**a**) feature and (**b**) weight memories, respectively, where ≪ stands for a barrel shifter.

Some remarks regarding other miscellaneous parts of the microarchitecture are followed.

- The other previous operation-skipping schemes are implemented as presented in our previous work [8]. The threshold-based and pooling-based operation-skipping schemes are implemented by controlling the counters to terminate the loop if the related conditions are satisfied. The boundary operation-skipping scheme is implemented by biasing the partial sum and changing the range of (u, v), as expressed by Lines 11 and 12 in Algorithm 1, respectively, for which dedicated components (the receptive-field range calculator and partial-sum bias calculator) are incorporated. The implementation details of these components are omitted here for brevity; the interested readers can be referred to our previous work.
- The partial sum is initialized by negating the value calculated by adding the threshold and bias that corresponds to the the receptive field size. The comparison of the partial sum to the threshold is thus efficiently implemented by picking the sign of the partial sum with no explicit comparison.
- Since the first block in the BCNN inference process usually has the input feature map with multi-bit elements, it has to be processed in the way that for the binary-weight network [2], differently from those for the other blocks. To support this kind of processing, the microarchitecture is designed so as to carry out the summation of the single-bit or multi-bit elements, resulting from the bitwise XNOR operations, as shown in Figure 5.
- The last block in the BCNN inference process produces the soft results, going to be used for the post-processing (e.g., calculating the class probabilities in the classification tasks). The summation result stored in the λ register corresponds to each of the soft results, which can be stored directly to the memory, as shown in Figure 5.

3.3. Prototype Inference System

A prototype inference system is developed in an FPGA by integrating all the essential components. Figure 7 shows the overall architecture of the inference system. TORRES is

implemented by integrating two cores, designed based on the processing pipelines shown in Figure 5. As in our previous work [9], the two cores in TORRES run the inference process cooperatively by computing the output feature map in opposite directions from each end until they meet at the same location. (One of the cores works according to the processing flow, opposite to that presented in Section 3.1; accordingly, it considers the window on the immediate right of the current window to find the spatial locality.) Though one of the cores computes more elements in the output feature map by skipping more operations than the other core; that is, processes more workload effectively, they are balanced in working time, as illustrated in Figure 7. The two cores share the memories, employed as scratch-pad buffers, storing the thresholds, features, and weights for processing part of a block. They have been implemented by employing dual-port block random-access memories (BRAMs).

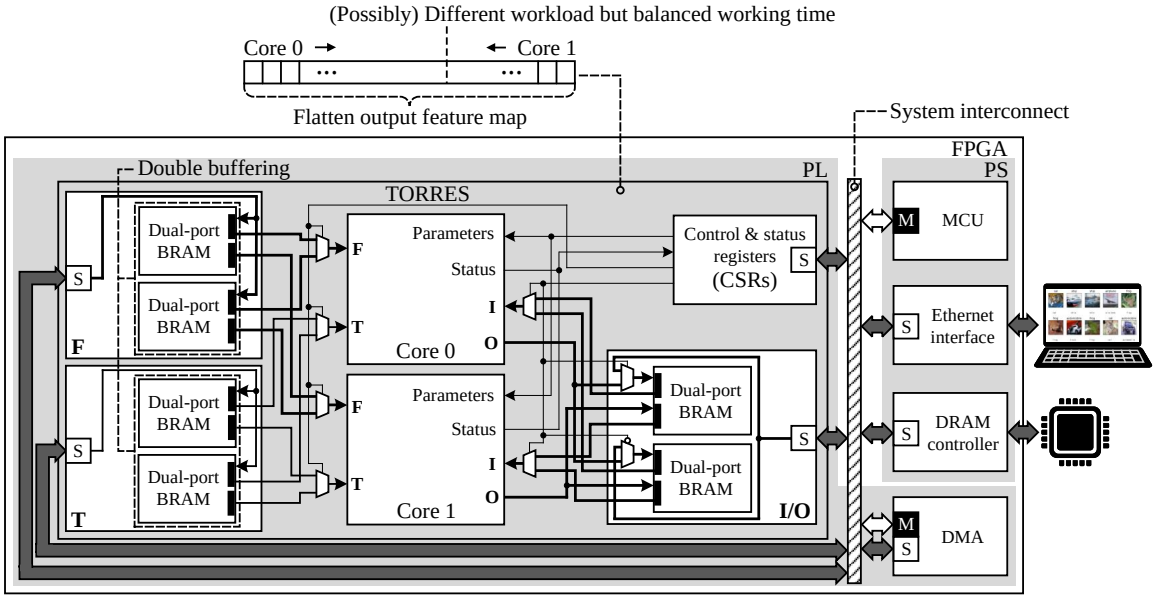


Figure 7. Prototype inference system, where PS, PL, M, and S stand for the MCU-based processing system, programmable logic, master, and slave interfaces, respectively.

The MCU fulfills controlling the components to realize the overall inference flow. The control and status registers (CSRs) as well as memories associated with TORRES are mapped to a region in the address space of MCU. A Linux operating system is ported to the MCU-based processing system, and a Python programming interface is provided to control the components, so that the functionality can be verified conveniently with high productivity. Through the web-based interactive interface, the verification is conducted from a remote computer.

The inference process is performed on a block-by-block basis, as illustrated in Figure 8. The n-th block is processed by being decomposed into M_n parts, where $n \in \{0, \cdots, N-1\}$, $M_n \geq 1$, and N is the number of the blocks. For processing each part, the direct memory access (DMA) loads the weights and thresholds, which are required for the processing, from the external dynamic random-access memory (DRAM) to the corresponding memories inside TORRES. Configured with the parameters related to tensor shapes and operation-skipping schemes, TORRES processes the part. The data loading by DMA can overlap in time with the processing by TORRES because the threshold and weight memories in TORRES are designed to support the double-buffering technique, as described in Figure 7. After processing every block, the inference process is finalized by producing the results stored in the feature map memory in TORRES, which may be used for the post-processing by MCU.

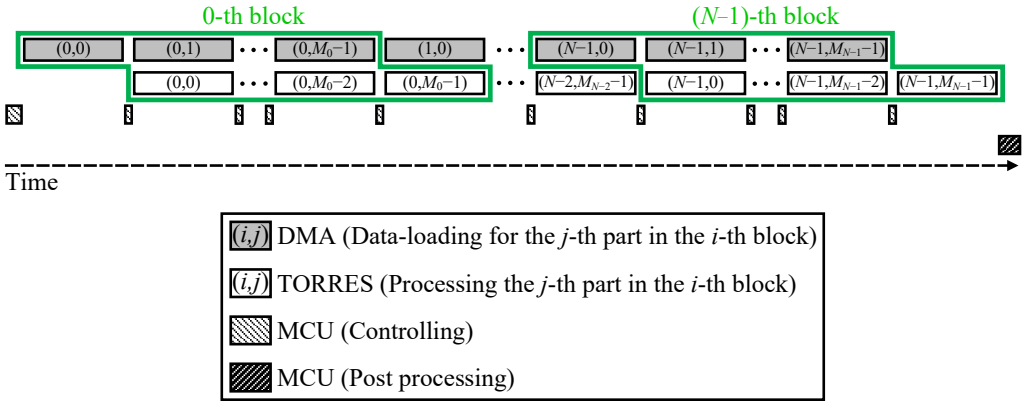


Figure 8. Overall flow for the BCNN inference process composed of N blocks, where the n-th block is processed by being decomposed into M_n parts.

4. Results and Evaluation

The prototype inference system based on TORRES has been synthesized for a 28 nm FPGA using Xilinx Vivado v2022.1. The entire system has been successfully fitted into a device; its resource usage is 11.8 K LUTs, 16.1 K flip-flops (FFs), and 384 K-bit BRAMs; the resource usage of TORRES itself is 2.31 K LUT, 0.9 K FFs, and 384 K-bit BRAMs, where one LUT corresponds to a six-input look-up table. No digital signal processors (DSPs) are used for synthesizing TORRES since even the MAC unit necessitated for the address generator for the weight memory has been implemented efficiently using LUTs because of the low bitwidth. (Note that DSPs are usually instantiated to synthesize general MAC units in FPGAs.) The maximum operating frequency and power consumption of TORRES for the synthesized design were estimated to be 211.1 MHz and 95 mW, respectively. Under the prototype inference system, the functionality of TORRES has been verified for practical inference tasks. Figure 9 shows the environmental setup for functional verification. A video demonstrating the functionality is provided through the webpage (https://abit.ly/torres (accessed on 13 October 2022)).

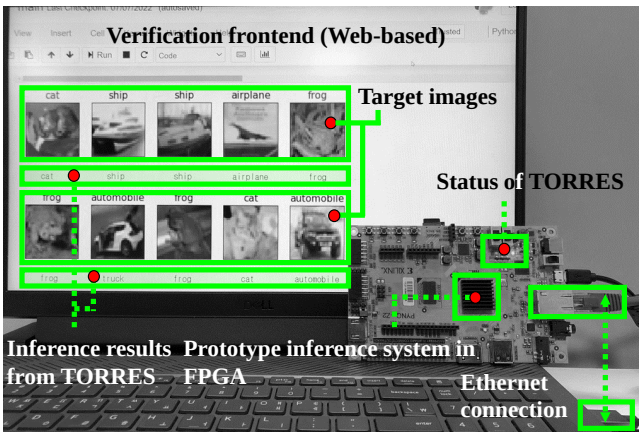
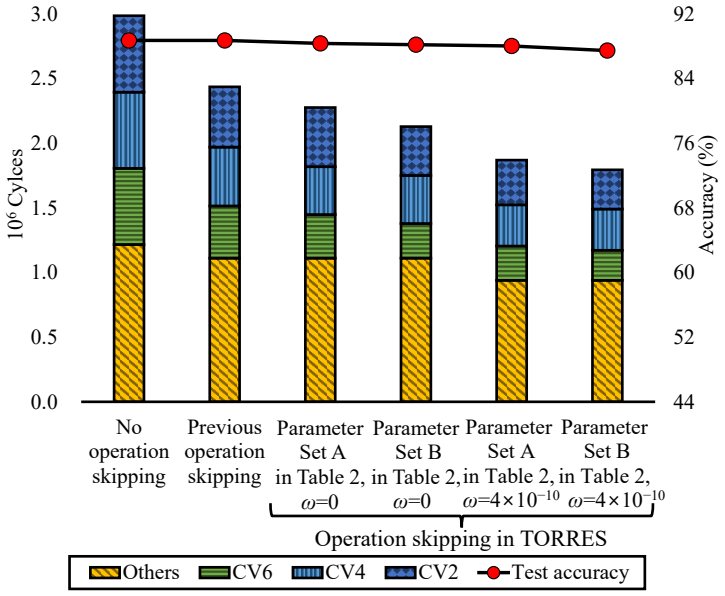


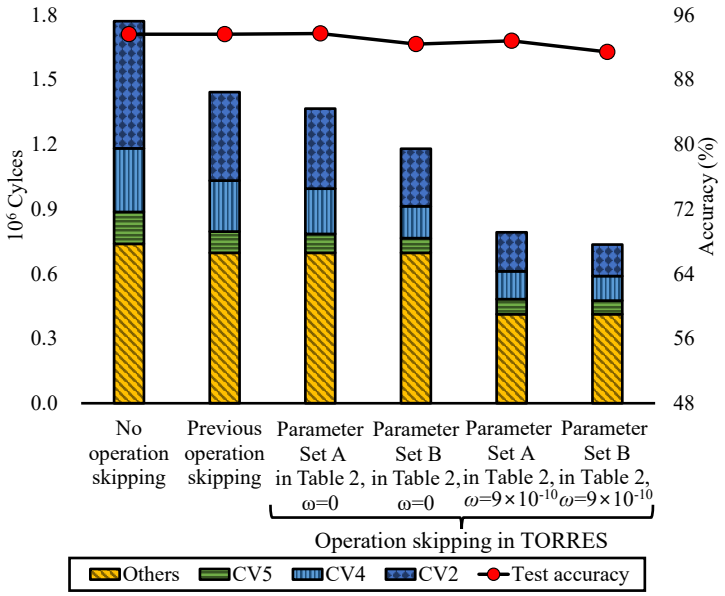
Figure 9. Environment setup for the functional verification.

The effects of the operation-skipping scheme proposed for TORRES have been analyzed for the practical inference tasks using CIFAR10 [25] and SVHN [26] datasets. (TORRES has been developed to perform the inference process based on XNOR-Net as stated in Section 2. Though XNOR-net is hardware-friendly as can be decomposed into several blocks of the consistent structure of binary-input and binary-output, it is known that the performance degradation from that achievable by the full-precision models is considerable in particular for the large-scale tasks [2]. Therefore, XNOR-Net is practically useful for the small-scale tasks; hence, most of the previous studies [7,8,13,14,17] presenting the efficient processors for

XNOR-net, evaluated the efficiency mainly for small-scale tasks, such as CIFAR10 classification task. Here, this study evaluates the performance of TORRES under the CIFAR10 as well as SVHN classification tasks.) Table 2 presents the BCNN models employed for these tasks. The proposed scheme can be applied by configuring the parameters (η and τ) for each block with pooling. The analysis has been conducted using the two parameter sets in Table 2, where Parameter Sets A and B are accuracy-oriented and operation-skipping-oriented, respectively. To find the effects more elaborately, the analysis has also been conducted without any operation-skipping scheme and with the previous schemes in [8]. Figure 10 presents the analysis results for the operation skipping schemes. The results are the averaged ones for the test data sets. The last two results in each figure have been obtained for the model trained with the proposed regularization technique with $\omega > 0$.



(**a**)



(**b**)

Figure 10. Cycle count and accuracy for the (**a**) CIFAR10 and (**b**) SVHN classification tasks. The second results in each figure have been obtained based on the previous operation skipping techniques [8].

Table 2. BCNN models and the parameters for the proposed operation skipping scheme used for the evaluation.

Target Task	Model Structure		Parameter Set A	Parameter Set B
	Block [a]	(w, h, k, c', s)	(η, τ)	(η, τ)
CIFAR10 classification [c]	CV1	(32, 32, 3, 128, 1)	-	-
	CV2	(32, 32, 3, 128, 2)	(1, 4) [b]	(1, 3)
	CV3	(16, 16, 3, 256, 1)	-	-
	CV4	(16, 16, 3, 256, 2)	(1, 3)	(1, 3)
	CV5	(8, 8, 3, 512, 1)	-	-
	CV6	(8, 8, 3, 512, 2)	(1, 3)	(1, 2)
	FC1	(1, 1, 1, 1024, 1)	-	-
	FC2	(1, 1, 1, 1024, 1)	-	-
	FC3	(1, 1, 1, 10, 1)	-	-
SVHN classification	CV1	(32, 32, 3, 128, 1)	-	-
	CV2	(32, 32, 3, 128, 2)	(2, 3)	(1, 2)
	CV3	(16, 16, 3, 128, 1)	-	-
	CV4	(16, 16, 3, 256, 2)	(2, 3)	(1, 2)
	CV5	(8, 8, 3, 256, 2)	(2, 3)	(2, 1)
	FC1	(1, 1, 1, 128, 1)	-	-
	FC2	(1, 1, 1, 10, 1)	-	-

[a] CVn and FCn stand for the n-th convolutional and fully-connected blocks, respectively. [b] The zero-prediction technique is not applied effectively by configuring τ to s^2. [c] The model for the CIFAR10 task has the same structure of that widely used for the evaluation of the BCNN inference processors in previous studies including [8,17].

The results in Figure 10 show that the proposed scheme skips operations significantly while maintaining the accuracy. In the figure, less cycle counts reflect more operations have been skipped. With Parameter Set B, the proposed scheme reduces the cycle counts taken to perform the CIFAR10 and SVHN classification tasks by up to 54.5% and 54.8%, respectively, and the reduction rates are further increased up to 60.5% and 75.2%, respectively, for the two tasks by the regularization technique. The overall cycle counts are reduced by 39.8% and 58.4% for the CIFAR10 and SVHN classification tasks, respectively. It can be seen that the cycle count reduction in the SVHN classification task is higher than that in the CIFAR10 classification task. This is because the ratio of the number of the blocks, which can be performed efficiently with the proposed locality-aware operation-skipping, to the total number of blocks is higher for the model employed to perform the SVHN task in this experiment; in addition, the regularization technique is applied with a higher scaling factor. The cycle count reduction by the proposed scheme is considerably high, even when assessed relatively to the cycle counts reduced by the schemes in [8]. The degradation in accuracy is negligible despite such a significant reduction in the cycle counts.

It is interesting to find the tradeoff between the cycle count and accuracy in Figure 10. When the parameter set is configured so as to be operation-skipping-oriented (Parameter Set B), the cycle counts are less than those obtained when the parameter set is configured so as to be accuracy-oriented (Parameter Set A); however, the accuracy degradation is slightly higher. More importantly, TORRES has been designed to provide such tradeoff between them with the configurability of the parameters in runtime. The inference speed and cycle counts are in inverse proportion, so are the efficiency and cycle counts.

The implementation results of TORRES are summarized in Table 3, along with those of the previous BCNN inference processors implemented in the FPGAs of the same technology node. TORRES shows a lot higher inference speed than our previous work [8]. Such a higher inference speed is attributed to not only the efficient processing flow based on the proposed operation-skipping scheme but also the efficient microarchitecture. It is notable that the resource usage of TORRES is not so higher than that of [8] as the proposed operation-skipping scheme has been implemented without much resource overhead. The processors presented in [7,13,14] show higher inference speeds than TORRES; however, they are not as resource-efficient as TORRES because their resource usages are much higher

than that of TORRES. The resource efficiency is a metric to show how fast the inference can be processed per each unit logic element in the implementation. This metric can be derived by the inference speed divided by the resource usage as presented in [7,8,13,14,17]. With the high inference speed and low resource usage, TORRES exhibits 1.45 times higher resource efficiency than the state-of-the-art BCNN inference processors. Notwithstanding the high resource efficiency, TORRES achieves comparable accuracy for the classification task.

Table 3. Implementation results of the BCNN inference processors in FPGAs.

BCNN Inference Processor	**TORRES [a] (with Param. Set A)**	**TORRES [a] (with Param. Set B)**	**[7]**	**[8]**	**[13]**	**[14]**	**[17]**
FPGA device [b] (Part number)	Zynq®-7000 (XC7Z020)	Zynq®-7000 (XC7Z020)	Zynq®-7000 (XC7Z020)	Cyclone®V (5CSXFC6D6)	Zynq®-7000 (XC7Z045)	Zynq®-7000 (XC7Z020)	Zynq®-7000 (XC7Z020)
Inference speed (GOP/s) [c]	255.2	291.2	722.0	83.0	13,389.0	329.0	208.0
Resource usage (KLUT) [d]	2.31	2.31	29.60	2.00	153.86	14.5	46.9
Energy eff. (TOP/J)	2.69	3.07	0.22	0.94	1.23	0.14	0.04
Resource eff. (MOP/s/LUT)	110.49	126.06	24.39	41.45	87.02	22.72	4.43
Classification acc. (%) [e]	88.04	87.47	88.61	88.88	88.70	81.80	88.18

[a] The parameter sets are described in Table 2. [b] All the devices used to implement the processors in the table are in the technology nodes of 28 nm, so that the direct comparisons of the results are viable. [c] Each binary operation has been counted by one OP. [d] Each six-input look-up table has been counted by one LUT. [e] This is the accuracy for the CIFAR10 classification task.

5. Conclusions

A resource-efficient processor for BCNN inference is proposed. The processing flow of the proposed processor is devised based on a novel operation-skipping scheme. Considering the spatial locality inherent in feature maps, the proposed scheme reorders the computation and predicts the results, thereby skipping some operations involved in computing the elements within the pooling windows. A regularization technique is also presented to skip more operations. The microarchitecture of the proposed processor is designed to actualize the proposed skipping scheme without a significant resource overhead. To efficiently generate addresses, the ordering types of the elements in the memories are carefully determined. A fully integrated inference system based on the proposed processor has been implemented in a 28 nm FPGA. Its functionality has been verified thoroughly for practical inference tasks. The resource efficiency of the proposed processor is as high as 126.06 MOP/s/LUT. To the best of our knowledge, this work is the first BCNN inference processor that exploits the spatial locality to skip operations. Further studies may follow this work to improve resource efficiency more significantly by enhancing how the spatial locality is found and how it is considered for skipping operations.

Author Contributions: Conceptualization, S.-J.L. and T.-H.K.; Data curation, S.-J.L. and G.-H.K.; Formal analysis, S.-J.L. and T.-H.K.; Funding acquisition, T.-H.K.; Investigation, S.-J.L. and G.-H.K.; Methodology, T.-H.K.; Project administration, T.-H.K.; Software, S.-J.L. and G.-H.K.; Supervision, T.-H.K.; Validation, S.-J.L. and G.-H.K.; Visualization, S.-J.L. and G.-H.K.; Writing—original draft, T.-H.K.; Writing—review & editing, S.-J.L., G.-H.K. and T.-H.K. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by ABOV Semiconductor [202200840001, Study of a micro NPU for a tiny MCU], Institute for Information & Communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) [2017-0-00528, The Basic Research Lab for Intelligent Semiconductor Working for the Multi-Band Smart Radar], and the GRRC program of Gyeonggi

province [2017-B02, Study on 3D Point Cloud Processing and Application Technology]. The EDA tools were supported by IDEC, Korea.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Hubara, I.; Courbariaux, M.; Soudry, D.; El-Yaniv, R.; Bengio, Y. Binarized neural networks. In Proceedings of the 30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain, 5–10 December 2016; pp. 4107–4115.
2. Rastegari, M.; Ordonez, V.; Redmon, J.; Farhadi, A. XNOR-Net: ImageNet classification using binary convolutional neural networks. In Proceedings of the European Conference Computer Vision, Amsterdam, The Netherlands, 11–14 October 2016; Springer: Berlin/Heidelberg, Germany, 2016; pp. 525–542.
3. Ferrarini, B.; Milford, M.J.; McDonald-Maier, K.D.; Ehsan, S. Binary Neural Networks for Memory-Efficient and Effective Visual Place Recognition in Changing Environments. *IEEE Trans. Robot.* **2022**, *38*, 2617–2631. [CrossRef]
4. Cerutti, G.; Cavigelli, L.; Andri, R.; Magno, M.; Farella, E.; Benini, L. Sub-mW Keyword Spotting on an MCU: Analog Binary Feature Extraction and Binary Neural Networks. *IEEE Trans. Circuits Syst. I* **2022**, *69*, 2002–2012. [CrossRef]
5. Conti, F.; Schiavone, P.D.; Benini, L. XNOR Neural Engine: A Hardware Accelerator IP for 21.6-fJ/OP Binary Neural Network Inference. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2018**, *37*, 2940–2951. [CrossRef]
6. Moons, B.; Bankman, D.; Yang, L.; Murmann, B.; Verhelst, M. BinarEye: An always-on energy-accuracy-scalable binary CNN processor with all memory on chip in 28 nm CMOS. In Proceedings of the Custom Integrated Circuits Conference, San Diego, CA, USA, 8–11 April 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 1–4.
7. Guo, P.; Ma, H.; Chen, R.; Li, P.; Xie, S.; Wang, D. FBNA: A Fully Binarized Neural Network Accelerator. In Proceedings of the 28th International Conference on Field Programmable Logic and Applications (FPL), Dublin, Ireland, 27–31 August 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 51–54.
8. Kim, T.H.; Shin, J. A resource-efficient inference accelerator for binary convolutional neural networks. *IEEE Trans. Circuits Syst. II* **2021**, *68*, 451–455. [CrossRef]
9. Kim, T.; Shin, J.; Choi, K. IOTA: A 1.7-TOP/J inference processor for binary convolutional neural networks with 4.7 K LUTs in a tiny FPGA. *IET Electron. Lett.* **2020**, *56*, 1041–1044. [CrossRef]
10. Geng, T.; Li, A.; Wang, T.; Wu, C.; Li, Y.; Shi, R.; Wu, W.; Herbordt, M. O3BNN-R: An out-of-order architecture for high-performance and regularized BNN inference. *IEEE Trans. Parallel Distrib. Syst.* **2021**, *32*, 199–213. [CrossRef]
11. Rasoulinezhad, S.; Fox, S.; Zhou, H.; Wang, L.; Boland, D.; Leong, P.H. MajorityNets: BNNs Utilising Approximate Popcount for Improved Efficiency. In Proceedings of the International Conference on Field-Programmable Technology (ICFPT), Tianjin, China, 9–13 December 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 339–342.
12. Scherer, M.; Rutishauser, G.; Cavigelli, L.; Benini, L. CUTIE: Beyond PetaOp/s/W ternary DNN inference acceleration with Better-than-Binary energy efficiency. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2022**, *41*, 1020–1033. [CrossRef]
13. Liu, Q.; Lai, J.; Gao, J. An Efficient Channel-Aware Sparse Binarized Neural Networks Inference Accelerator. *IEEE Trans. Circuits Syst. II* **2022**, *69*, 1637–1641. [CrossRef]
14. Nakahara, H.; Fujii, T.; Sato, S. A fully connected layer elimination for a binarized convolutional neural network on an FPGA. In Proceedings of the 27th International Conference on Field Programmable Logic and Applications (FPL), Gent, Belgium, 4–6 September 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 1–4.
15. Liu, Z.; Shen, Z.; Savvides, M.; Cheng, K.T. Reactnet: Towards precise binary neural network with generalized activation functions. In Proceedings of the European Conference Computer Vision, Glasgow, UK, 23–28 August 2020; Springer: Berlin/Heidelberg, Germany, 2020; pp. 143–159.
16. Lin, X.; Zhao, C.; Pan, W. Towards accurate binary convolutional neural network. In Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA, 4–9 December 2017; pp. 345–353.
17. Zhao, R.; Song, W.; Zhang, W.; Xing, T.; Lin, J.H.; Srivastava, M.; Gupta, R.; Zhang, Z. Accelerating binarized convolutional neural networks with software-programmable FPGAs. In Proceedings of the International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 22–24 February 2017; ACM: New York, NY, USA, 2017; pp. 15–24.
18. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. In Proceedings of the International Conference Learning Representations, San Diego, CA, USA, 7–9 May 2015; pp. 1–14.
19. Umuroglu, Y.; Fraser, N.J.; Gambardella, G.; Blott, M.; Leong, P.; Jahre, M.; Vissers, K. FINN: A framework for fast, scalable binarized neural network inference. In Proceedings of the International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 22–24 February 2017; ACM: New York, NY, USA, 2017; pp. 65–74.
20. Sze, V.; Chen, Y.H.; Yang, T.J.; Emer, J.S. Efficient processing of deep neural networks: A tutorial and survey. *Proc. IEEE* **2017**, *105*, 2295–2329. [CrossRef]

21. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. Pytorch: An imperative style, high-performance deep learning library. In Proceedings of the 33rd International Conference on Neural Information Processing Systems, Vancouver, BC, Canada, 8–14 December 2019; pp. 1–12.
22. Weng, J.; Jain, A.; Wang, J.; Wang, L.; Wang, Y.; Nowatzki, T. UNIT: Unifying tensorized instruction compilation. In Proceedings of the International Symposium on Code Generation and Optimization, Seoul, Korea, 27 February–3 March 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 77–89.
23. Zhou, K.; Tan, G.; Zhang, X.; Wang, C.; Sun, N. A performance analysis framework for exploiting GPU microarchitectural capability. In Proceedings of the International Conference Supercomputing, Chicago, IL, USA, 14–16 June 2017; pp. 1–10.
24. Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; et al. TensorFlow: A system for Large-Scale machine learning. In Proceedings of the USENIX Symposium on Operating Systems Design and Implementation, Savannah, GA, USA, 2–4 November 2016; pp. 265–283.
25. Krizhevsky, A.; Hinton, G. *Learning Multiple Layers of Features from Tiny Images*; Technical Report; Citeseer: University Park, PA, USA, 2009.
26. Netzer, Y.; Wang, T.; Coates, A.; Bissacco, A.; Wu, B.; Ng, A.Y. Reading digits in natural images with unsupervised feature learning. In Proceedings of the NIPS Workshop on Deep Learning & Unsupervised Feature Learning, Granada, Spain, 12–17 December 2011; NeurIPS Foundation: Grenada, Spain, 2011; pp. 1–9.