PAPER

FCA-BNN: Flexible and Configurable Accelerator for Binarized Neural Networks on FPGA

Jiabao GAO[†a)], ***Member***, **Yuchen YAO**[†], **Zhengjie LI**[†], ***Nonmembers***, ***and*** **Jinmei LAI**[†b)], ***Member***

SUMMARY A series of Binarized Neural Networks (BNNs) show the accepted accuracy in image classification tasks and achieve the excellent performance on field programmable gate array (FPGA). Nevertheless, we observe existing designs of BNNs are quite time-consuming in change of the target BNN and acceleration of a new BNN. Therefore, this paper presents FCA-BNN, a flexible and configurable accelerator, which employs the layer-level configurable technique to execute seamlessly each layer of target BNN. Initially, to save resource and improve energy efficiency, the hardware-oriented optimal formulas are introduced to design energy-efficient computing array for different sizes of padded-convolution and fully-connected layers. Moreover, to accelerate the target BNNs efficiently, we exploit the analytical model to explore the optimal design parameters for FCA-BNN. Finally, our proposed mapping flow changes the target network by entering order, and accelerates a new network by compiling and loading corresponding instructions, while without loading and generating bitstream. The evaluations on three major structures of BNNs show the differences between inference accuracy of FCA-BNN and that of GPU are just 0.07%, 0.31% and 0.4% for LFC, VGG-like and Cifar-10 AlexNet. Furthermore, our energy-efficiency results achieve the results of existing customized FPGA accelerators by 0.8× for LFC and 2.6× for VGG-like. For Cifar-10 AlexNet, FCA-BNN achieves 188.2× and 60.6× better than CPU and GPU in energy efficiency, respectively. To the best of our knowledge, FCA-BNN is the most efficient design for change of the target BNN and acceleration of a new BNN, while keeps the competitive performance.

key words: *BNN, FPGA accelerators, hardware-oriented optimal formulas, analytical mode, mapping flow*

1. Introduction

Field programmable gate arrays (FPGAs) offer high flexibility, performance and energy-efficiency, thus there is a lot of attention in FPGA for accelerating convolutional neural networks (CNNs) [1]–[5]. Nevertheless, most designs of CNNs generally have sub-millisecond latency, high power consumption and low energy-efficiency. To overcome the above issues, Binarized Neural Networks (BNNs) [6] on FPGAs have recently attracted a growing attention [7]–[12]. Fasfous et al. [13] provide a solution for accurate BNN [14], which employs DSPs to execute the multiplications and XNOR-Popcount operations concurrently, but they just give the evaluation of processing element without realization of a whole network. Moreover, the computing performance is bound by limited DSP resource in FPGAs [15]. Therefore, FPGAs with abundant LUT resources are ideal for accelerating BNNs with only XNOR-Popcount operations to better support real-time embedded applications with acceptable accuracy [7].

However, for efficiently switching BNN models with respect to different requirements and accelerating a new BNN on mobile platforms, a framework for building FPGA accelerators to the corresponding BNN models is worth researching. Recently, there are already some frameworks proposed for building the corresponding FPGA accelerators for different models [16]–[19]. To accelerate different target models, these authors need to call the corresponding code templates of required operations to construct the computing arrays with respect to the design parameters, and then implement corresponding accelerators. Firstly, according to structures of SFC, LFC and VGG-like models, and their respective performance requirements, FINN proposed in [16] first determines scale parameters to describe computing units of each layer, and then employs these parameters to generate different accelerators for each BNN model. Secondly, FP-BNN proposed in [17] employs design parameters of tiling and scheduling to implement the respective hardware accelerators for MLP, VGG-like, and AlexNet models. Thirdly, the design method of ReBNet [18] is similar to that of both FINN and FP-BNN, which allows users to specify the parallelism computing factors of each layer, and then generates the corresponding accelerators to SLC, VGG-like and AlexNet models. Finally, different from FINN, FP-BNN and ReBNet, LUTNet [19] directly utilizes the basic resource LUTs to implement accelerator design. Moreover, LUTNet uses the self-design implementation flow of FPGA to generate the corresponding accelerators to LFC, VGG-like and AlexNet models. In short, the existing works need to at least load the bitstream into the FPGA while switch of the target model, and even generate the bitstream to a new target model. Nevertheless, loading bitstream can affect the efficiency of the target model change, and generating bitstream is the very time-consuming process [16]–[19], which usually takes at least several hours for large-scale design. Therefore, it is worth of exploring a flexible and configurable accelerator, which can switch the target BNNs quickly for different application and reprogram to the target FPGA rapidly after each improvement of BNNs.

Under the premise of ensuring that the accelerator can rapidly complete switch of the target model and acceleration of a new model, the improvements of inference accuracy and performance are also crucial.

Firstly, to minimize the differences of inference

Manuscript received March 9, 2021.
Manuscript publicized May 19, 2021.
[†]The authors are with State Key Laboratory of ASIC and System, Fudan University, Shanghai, 201203, China.
a) E-mail: 17112020018@fudan.edu.cn
b) E-mail: jmlai@fudan.edu.cn (Corresponding author)
DOI: 10.1587/transinf.2021EDP7054

accuracy (***DoIA***) between on FCA-BNN and on GPU, many works employ padding operations to input feature maps (IFM) in CNOV layers, achieving small ***DoIA*** [12], [16], [17]. However, the computing arrays of existing designs are just needed to complete a specific model with the certain sizes of padded CONV operations to reduce ***DoIA***.

Secondly, to achieve required performance at minimal hardware resource cost, the corresponding analytical models are proposed to determine the design parameters for their accelerators [16], [17]. Umuroglu et al. [16] develop the customized roofline model to their accelerator framework by using the methodology in [20], and obtain required performance for LFC and VGG-like models; Liang et al. [17] also propose their customized analytical model, and use this model to estimate resource cost, determine the size of task tiles, then get remarkable performance for MPL, VGG-like and AlexNet models. However, the existing analytical models are not applicable to our goal to accelerate multiple models on a uniform accelerator.

To sum up, there are several challenges to design a flexible and configurable accelerator at reasonable resource cost. Firstly, the computing array needs to calculate different sizes of padded CONV and FC operations while still using 1 bit to represent activations ±1, padding value 0 and invalid output, which makes the arrays maintain binary design while achieving small ***DoIA***. Secondly, the analytical model needs to determine the optimal design parameters which make different BNN models accelerate on a uniform accelerator efficiently. Thirdly, the process of the mapping flow for switching target network and accelerating a new network needs to be quick. To overcome these challenges, we propose FCA-BNN accelerator with its corresponding analytical model and mapping flow. Specifically, the main contributions of this work are as follows:

- Efficient adaptivity to different sizes of padded CONV and FC operations. By using the character of Popcount operations, the original formulas are transformed into the hardware-friendly formulas to design the energy-efficient computing arrays, achieving binary design and small ***DoIA***.
- Efficient acceleration for most popular BNN models. The analytical model is proposed to explore the optimal design parameters for the uniform computing arrays. This makes FCA-BNN achieve the competitive results in terms of performance, and energy efficiency.
- Efficient mapping to change of the target BNN and acceleration of a new target BNN. The mapping flow is developed to switch the target BNN by just entering order and accelerate a new target BNN by compiling and loading instructions, without loading and generating bitstream.
- Combining all the contributions above, we implement FCA-BNN on XC7Z100 FPGA and evaluate it for LFC, VGG-like, Cifar-10 AlexNet. The results show these BNNs on FCA-BNN achieve small ***DoIA*** and high performance.

The rest of this paper is organized as follows: Sect. 2 proposes the hardware-oriented optimization formulas for FCA-BNN. The corresponding hardware accelerator design is presented in Sect. 3. The analytical model and mapping flow both for FCA-BNN are introduced in Sects. 4 and 5, respectively. Experimental results will be discussed in Sect. 6, and conclusion will be given in Sect. 7.

2. Hardware-Oriented Optimization Formulation

2.1 BNN Inference

Since BNN is an extreme version of CNN, its construct is also a stack of CONV layers followed by a Pooling layer, and FC layers. However, benefited from the weights and activizations constrained to +1 and −1, BNNs can replace the MAC operations with the XNOR-Popcount operations. Hence, the calculation process of CONV and FC is shown as follows:

$$\begin{aligned} p_{l,n,r,c} &= \sum_{m=0}^{IC_l-1} \sum_{j=-\left\lfloor \frac{K_l}{2} \right\rfloor}^{K_l-\left\lceil \frac{K_l}{2} \right\rceil} \sum_{i=-\left\lfloor \frac{K_l}{2} \right\rfloor}^{K_l-\left\lceil \frac{K_l}{2} \right\rceil} (w_{l,n,m,j,i} \odot f_{l-1,m,r+j,c+i}) \\ y_{l,n,r,c} &= 2p_{l,n,r,c} - L_l + bias_{l,n} \\ f_{l,n,r,c} &= sign(bn(y_{l,n,r,c})) \end{aligned} \tag{1}$$

Where $\odot$ represents XNOR logic operation; *bn* and *sign* are batch normalization (BN) and binarization (BIN) functions, respectively. The meanings of the subscripts in Eq. (1): l represents the l^{th} layer of BNN; r and c denote the r^{th} row and the c^{th} column in the output feature map (OFM); m and n denote the m^{th} input and the n^{th} output channels. The meanings of the variables in Eq. (1): IC and K represent the number of input channels and the kernel size; L and *bias* are the count of weights used for computing one OFM and the bias value, respectively; f and w are the feature map and weights; p is the sum of number of +1 s which output from $w \odot f$; y is the result of convolution. Note that K is equal to odd numbers, i.e., 1, 3, 5, 7 and 11 in most network models [6], [16], [21], [22].

2.2 XNOR-Popcount Optimized to FCA-BNN

As shown in Eq. (1), the CONV and FC layers insist of a large number of XNOR-Popcount operations, however, FPGAs have limited resources. Hence, these operations are divided into tiles of a reasonable size by the loop tiling technique of FPGA hardware design [12], [16], [17]. Assume the computing array (Sect. 3.3) has N PEs for N parallel output channels. Moreover, each PE, including H XNOR engines, H Adder trees and H Accumulators (ACCU), is responsible for calculating the outputs of H columns of the same raw in one channel. Each XNOR engine has W XNOR elements for one output. Hence, the parameters of maximum possible parallelism can be expressed as follows:

$$\begin{aligned} PIC_m &= \left\lfloor \log_2\left(\frac{W}{K_l}\right) \right\rfloor \\ PICOL_m &= (H-1) * S_l + K_l \\ POC_m &= N \end{aligned} \tag{2}$$

$$POCOL_m = H$$

Here, S is the stride of the convolution; PIC_m is the max count of parallel input channels which is generally equal to a powers-of-2 positive integer in most layers except the first layer [6], [16], [21], [22]; $PICOL_m$ is the max count of parallel input columns; POC_m and $POCOL_m$ are the max count of parallel output channels and columns. Besides, the numbers of parallel input and output rows both equal to 1.

To match with the computing array, we develop a Channel-Column-Tile partition (CCTP) pattern, which broadcasts $PIC_m * PICOL_m$ inputs to each PE at each cycle. Hence, when the l^{th} layer has more input channels than the array can handle, it is divided into $T_{ic} = \left\lceil \frac{IC_l}{PIC_m} \right\rceil$ tiles. After finishing one tile, the integer variable t_i increases one, which ranges from 0 to $T_{ic} - 1$.

Specifically, $PICOL_m$ inputs of each channel are first partitioned into H groups by the rule with K_l consecutive inputs and S_l stride. Afterwards, K_l consecutive inputs with the same locations from PIC_m input channels in each tile are regrouped into H input blocks, and then the blocks are separately brought into H XNOR engines. Consequently, the results from the XNOR operations are fed into Adder tree to calculate $pt_{l,n,r,c}$ which accumulates for $T_{ic} * K_l$ cycles in ACCU to produce the POP result $ph_{l,n,r,c}$. Hence, $pt_{l,n,r,c}$ and $ph_{l,n,r,c}$ are calculated as follows:

$$\begin{aligned} pt_{l,n,r,c} &= \sum_{m=t_i*PIC_m}^{(t_i+1)*PIC_m-1} \sum_{i=-\lfloor \frac{K_l}{2} \rfloor}^{K_l-\lceil \frac{K_l}{2} \rceil} (w_{l,n,m,j,i} \odot f_{l-1,n,m,r+j,c+i}) \\ t_i &= (0, 1, \ldots, T_{ic} - 1) \\ ph_{l,n,r,c} &= \sum_{t_i=0}^{T_{ic}-1} \left(\sum_{j=-\lfloor \frac{K_l}{2} \rfloor}^{K_l-\lceil \frac{K_l}{2} \rceil} (pt_{l,n,r,c}) \right) \end{aligned} \tag{3}$$

However, to simultaneously meet different kernel sizes of CONV and FC, W XNOR elements and $PIC_m * K_l$ inputs satisfy the inequation $W \geq PIC_m * K_l$. Suppose there are $XNOR_{idle}$ idle XNOR elements in the current operations. Hence, the relationship between W, $PIC_m * K_l$ and $XNOR_{idle}$ can be expressed as:

$$W = PIC_m \times K_l + XNOR_{idle}, \quad XNOR_{idle} \geq 0 \tag{4}$$

Meanwhile, to reduce ***DoIA***, zero-padding is used to IFM, which means adding zeros around the outside of the IFM when either $r + j$ or $c + i$ is smaller than 0. Obviously, since the idle XNOR elements and zero-padding are introduced, the hardware design needs at least two bits to represent +1, −1 and 0, which leads to high resource cost. However, we observe that Popcount operations are used to count the number of +1 s from XNOR operations. Inspired by this observation, we use a set bit (1) and an unset bit (0) to represent +1 and $\{-1, 0\}$, respectively, which implements binary design. Besides, we use the two functions, reset and clock enable, of flip-flop to implement the control of idle elements and zero-padding, respectively, which means the elements do not use additional LUT resource. Therefore, the calculation of $pt_{l,n,r,c}$ can be transformed as follow:

$$\begin{aligned} pt'_{l,n,r,c} = &\sum_{m=t_i*PIC_m}^{(t_i+1)*PIC_m-1} \sum_{i=-\lfloor \frac{K_l}{2} \rfloor}^{K_l-\lceil \frac{K_l}{2} \rceil} \left((w_{l,n,m,j,i} \odot f_{l-1,n,m,r+j,c+i}) \cdot \bar{P} \right) \\ &+ \sum_{id=0}^{XNOR_{idle}-1} v_{id}|_{clk\_EN=\bar{I}} \end{aligned} \tag{5}$$

Where $\bar{P}$ and $\bar{I}$ are the reset and clock enable signals of flip-flop, respectively; v_{id} is the output from the id^{th} idle XNOR element. More specifically, when $\bar{P}$ is active, the XNOR element output 0 with no regard for f and w, meaning that 0-padding enable is valid; When $\bar{I}$ is active, the clock transition of the XNOR element is ignored, meaning that the idle elements can save energy. Accordingly, the POP result $ph'_{l,n,r,c}$ can be got by accumulating $pt'_{l,n,r,c}$.

To sum up, we employ hardware-oriented optimization formula to design energy-efficient computing array which efficiently supports different sizes of padded CONV and FC operations. More importantly, it keeps binary design with saving LUT resource and improving energy efficiency.

2.3 Max Pooling (MP) Optimized to FCA-BNN

In most popular BNNs [6], [16], [21], [22], 2×2 and 3×3 MPs both with a stride of 2 are the most commonly used MP operations. However, to support the two MPs concurrently, the hardware design at least needs 4 3-input comparators. To reduce the resource cost, we propose the three-stage-comparison (TSCMP) engine shown as follows:

$$\begin{aligned} &\textbf{\textit{Stage}}\ \mathbf{1} \text{->} M1'_{l,n,r,c} = max(ph'_{l,n,r,c}, ph'_{l,n,r,c+1}, ph'_{l,n,r,c+2}) \\ &M2'_{l,n,r,c} = max(ph'_{l,n,r+1,c}, ph'_{l,n,r+1,c+1}, ph'_{l,n,r+1,c+2}) \\ &\textbf{\textit{Stage}}\ \mathbf{2} \text{->} M12'_{l,n,r,c} = max(M1'_{l,n,r,c}, M2'_{l,n,r,c}) \\ &M3'_{l,n,r,c} = max(ph'_{l,n,r+2,c}, ph'_{l,n,r+2,c+1}, ph'_{l,n,r+2,c+2}) \\ &\textbf{\textit{Stage}}\ \mathbf{3} \text{->} pm'_{l,n,r,c} = max(M12'_{l,n,r,c}, M3'_{l,n,r,c}) \end{aligned} \tag{6}$$

Where, max is the function for taking the maximum value. For 3×3 MP: In stage 1, 2 3-input comparators are used to take the max values $M1'_{l,n,r,c}$ and $M2'_{l,n,r,c}$ from the POP outputs of the first and second rows, respectively; In stage 2, the 2-input comparator is used to get the max value $M12'_{l,n,r,c}$ of between $M1'_{l,n,r,c}$ and $M2'_{l,n,r,c}$. Meanwhile, one of the 2 3-input comparators is used to take the max value $M3'_{l,n,r,c}$ from POPs of the third row; In stage 3, another 2-input comparator is used to get the max value $pm'_{l,n,r,c}$ of between $M12'_{l,n,r,c}$ and $M3'_{l,n,r,c}$. Additionally, for 2×2 MP, the TSCMP is just needed the first two stages to take the max value, and then output it to BN engine by pipeline in stage 3.

To sum up, the TSCMP just insists of 2 3-input and 2 2-input comparators, which means it uses lower resource cost to efficiently support 2×2 and 3×3 MPs. Note that for the CONV layer without MP, the $pm'_{l,n,r,c}$ is equal to the

POP output $ph'_{l,n,r,c}$, meaning that the MP is bypassed.

2.4 Batch Normalization (BN) Optimized to FCA-BNN

For the output $pm'_{l,n,r,c}$ from MP, there are three extra calculation phases to calculate the input $f_{l,n,r,c}$ of the next layer. Firstly, the $pm'_{l,n,r,c}$ is converted into the result $y'_{l,n,r,c}$. Secondly, the $y'_{l,n,r,c}$ is normalized in BN which is compulsory for high accuracy in BNNs [8], [9]. Finally, the output $x'_{l,n,r,c}$ from BN is binarized in BIN by comparing with 0. This calculation process is shown as follows:

$$\begin{aligned} &\boldsymbol{CONV}{->}y'_{l,n,r,c} = 2pm'_{l,n,r,c} - L_l + bias_{l,n} \\ &\boldsymbol{BN}{->}x'_{l,n,r,c} = \frac{y'_{l,n,r,c} - \mu}{\sqrt{\sigma^2}}\gamma + \beta \\ &\boldsymbol{BIN}{->}f_{l,n,r,c} = sign(x'_{l,n,r,c}) = \begin{cases} +1 & x'_{l,n,r,c} \geq 0 \\ -1 & x'_{l,n,r,c} < 0 \end{cases} \end{aligned} \tag{7}$$

Obviously, BN is the complex operation shown in Eq. (7). To reduce the cost of the hardware design, the BN together with BIN is transformed into the threshold-based comparison [8], [10], [16]. Furthermore, to better keep the accuracy of the output, we propose an optimization which considers bias in the threshold-based comparison. This calculation process is as follows:

$$\begin{aligned} &x'_{l,n,r,c} = \frac{y'_{l,n,r,c} - \mu}{\sqrt{\sigma^2}}\gamma + \beta = 0 \\ &\Rightarrow th_{l,n} = \frac{1}{2} \times \left(L_l - bias_{l,n} + \mu - \frac{\sqrt{\sigma^2}}{\gamma} \times \beta \right) \\ &\Rightarrow f_{l,n,r,c} = \begin{cases} +1 & pm'_{l,n,r,c} \geq th_{l,n} \\ -1 & pm'_{l,n,r,c} < th_{l,n} \end{cases} \end{aligned} \tag{8}$$

However, when $f_{l,n,r,c}$ is not needed to binarized in the final layer, the above optimization is no longer adapted for this condition. Hence, linear operation including bias is proposed, which replaces the BN along with the bias. This replacing process is as follows:

$$\begin{aligned} &f_{l,n,r,c} = x'_{l,n,r,c} = \frac{y'_{l,n,r,c} - \mu}{\sqrt{\sigma^2}}\gamma + \beta \\ &f_{l,n,r,c} = \frac{(2pm'_{l,n,r,c} - L_l + bias_{l,n}) - \mu}{\sqrt{\sigma^2}}\gamma + \beta \\ &f_{l,n,r,c} = A \times pm'_{l,n,r,c} + B \\ &\Rightarrow A = \frac{2\gamma}{\sqrt{\sigma^2}},\ B = \beta + \frac{(bias_{l,n} - L_l - \mu) \times \gamma}{\sqrt{\sigma^2}} \end{aligned} \tag{9}$$

For Eq. (8), $ph'_{l,n,r,c}$ is directly compared with threshold $th_{l,n}$. For Eq. (9), $pm'_{l,n,r,c}$ needs to multiply the scale parameter A, and then plus the bias parameter B.

3. Accelerator Architecture

3.1 Overview

Figure 1 shows an overview of the BNN inference accelerator. When FCA-BNN starts accelerating the target BNN, Global Controller controls the schedule of the following phases to complete acceleration of the target BNN. During a load data instruction, the image data and trained parameters are transferred from Off-chip Memory (DDR3) to Feature Map Buffers (FMB) and Param Buffers (PB) blocks, respectively. During a compute acceleration instruction, Layer-Level Configurable Registers (LLCR) block (Sect. 3.2) based control configures Computing Acceleration for BNN (CA-BNN), including Computing Arrays (Sect. 3.3), Pooling (Sect. 3.4) and Batchnorm (Sect. 3.5), as the required type of macro layer, and then enables the CA-BNN to execute seamlessly each layer of the target BNN by pipeline mode. Moreover, during inference, the CA-BNN takes inputs and trained parameters from FMB and PB (Sect. 3.6), respectively, to output results which are stored in FMB for the computing of the next macro layer.

For change of a target BNN among the existing BNNs, the mapping flow (Sect. 5) is just needed to enter the corresponding order, and then FCA-BNN performs the above phases. For acceleration of a new BNN, the mapping flow is needed to compile the corresponding instructions and data for the new BNN. Afterwards, by the communication between PC and ARM-based Processor, the instructions and data are loaded into LLCR block and DDR3, respectively. Finally, FCA-BNN starts accelerating the new BNN.

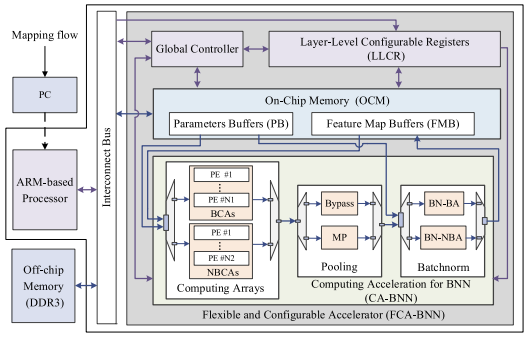

Fig. 1 The overview of computing acceleration.

Table 1 Structures of five types of macro layers.

Type of macro layer	Structure
CONV/FC (Generally for the first Layer)	NBCAs-(BN-BA)
CONV (Generally for the first Layer)	NBCAs-MP-(BN-BA)
CONV (Generally for the hidden Layers)	BCAs-MP-(BN-BA)
CONV/FC (Generally for the hidden Layers)	BCAs-(BN-BA)
FC (Generally for the last Layer)	BCAs-(BN-NBA)

3.2 Layer-Level Configurable Registers (LLCR) Design

To be compatible with the structures of most popular BNNs, the LLCR block stores the instructions which are used to configure and enable the CA-BNN to different types of macro layers shown in Table 1. Hence, FCA-BNN executes seamlessly operations of each layer to complete the target BNN. One instruction of each macro layer consists of the following signals.

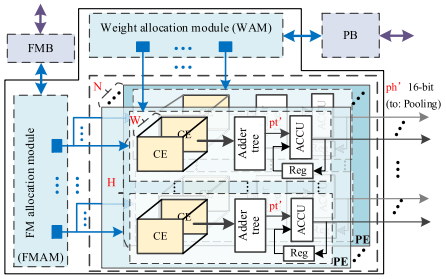

Fig. 2 Architecture of BCAs and NBCAs. Purple double arrows represent the instructions and control from LLCR and Global Controller, respectively.

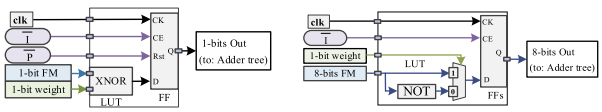

Fig. 3 (a) XNOR element; (b) NOT element.

FCA_En is used to enable Global Controller and start CA-BNN. **IFM_Type** is used to set either BCAs or NBCAs in Computing Arrays as the active array. **CONV_FC** is used to configure the active array as either CONV or FC operations. **IFM_Size** and **Kernel_Size** are the IFM and kernel sizes of CONV layers, respectively. **MP_En** denotes whether or not MP operation is needed. **MP_Size** stands for the size of MP. **BN_Type** is used to set either BN-BA or BN-NBA in Batchnorm block as the active engine. **Layer_Done** and **Net_Done** denote the current macro layer done and the network done, respectively.

3.3 Energy-Efficient Computing Arrays Design

As shown in Fig. 1, the Computing Array block consists of BCAs and NBCAs which are used to perform both padded CONV and FC operations for binary input and signed fixed-point input, respectively. Hence, BCAs and NBCAs employ the same architecture of computing array except computing element (CE) of PE which is used to calculate different types of inputs.

Figure 2 shows the architecture of BCAs and NBCAs. As stated in Sect. 2.2, N PEs are responsible for the XNOR-Popcount operations of N parallel output channels in one tile. FMAM and WAM employ the CCTP pattern and its corresponding weights allocation technique, respectively. For operations of each layer, initially, either BCAs or NBCAs accepts the instruction (including **IFM_Type**, **IFM_Size** and **Kernel_Size** signals) and the control from LLCR and Global Controller. Afterwards, FMAM and WAM fetch the inputs and weights from FMB and PB, and then allocate them to the corresponding PEs. Consequently, the outputs are passed to Pooling block.

For CE (XNOR element) used to calculate binary input in BCAs, we adopt Eq. (5) to design it shown in Fig. 3 (a), which insists of an XNOR gate and a FF. When the element executes padding operation, its output is always equal to "0", otherwise its output is equal to the result from XNOR operation between FM and weight. Additionally, for CE (Not element) used to calculate fixed-point input in NBCAs, it insists of one 8-bit NOT gate and one 2-input MUX, as shown in Fig. 3 (b). When the weight value is 1, the output is equal to the input, otherwise the output is equal to the opposite of the input including padding bit 0. Furthermore, clock enable signal $\bar{I}$ is invalid except that of the activating CEs to save energy, meaning that the computing arrays can improve the overall energy efficient.

For CONV operations, the PEs adopt CCTP. Hence, $PIC_m * PICOL_m$ inputs and $POC_m * PIC_m * K_l$ weights are brought into the PEs to calculate the partial results of $POC_m * POCOL_m$ POPs. For FC operations, since there are just two dimensions including input and output neurons, W XNOR elements of each XNOR engine are used to receive PIN_m inputs and PIN_m weights, and then $N * H$ engines produce the partial results of PON_m POPs. Therefore, as shown in Eq. (10), the PEs take T_{CONV} and T_{FC} cycles to finish the CONV and FC operations, respectively, by Eq. (2).

$$T_{CONV} = \frac{NOC}{POC_m} * \frac{NC * NR * NIC * K_l * K_l}{POCOL_m * PIC_m * K_l}$$
$$T_{FC} = \frac{NON}{PON_m} * \frac{NIN}{PIN_m}, \ (PON_m \leq N * H, \ PIN_m \leq W) \tag{10}$$

Here, NOC, NIC, NR and NC denote the number of the output and input channels, and the rows and columns of the IFM, respectively, in CONV layer. NIN and NON represent the input and output neurons, respectively, in FC layer.

Furthermore, to detail how FMAM and WAM allocate inputs and weights to $N \times H \times W$ XNOR elements for the computing of different sizes of the padded CONV and FC operations. Suppose $N = 1$, $H = 2$, and $W = 6$. For CONV layer with $NOC = 2$, $NIC = 2$, $NR = NC = 2$, $K_l = 5$, and $S_l = 1$, we get $POC_m = 1$, $PIC_m = 1$, $POCOL_m = 2$, $PICOL_m = 6$ and $XNOR_{idle} = 1$ by Eqs. (2) and (4). As illustrated in Fig. 4 (a), from 1/11 to 10/20 cycles, the PE can generate POPs of two columns of the first row in the first and second output channels, respectively; from 21/31 to 30/40 cycles, other two POPs of the second row in the two output channels can be generated. During the process, the XNOR element produces 0 when it receives padding enable signal (P_i). Moreover, $XNOR_{idle} = 1$ means the clock enable signal of one XNOR element in each engine is always invalid in the whole process. Similarly, for CONV layer with the same parameters except $K_l = 3$, we can get $POC_m = 1$, $PIC_m = 2$, $POCOL_m = 2$, $PICOL_m = 4$ and $XNOR_{idle} = 0$, and thus the PE takes 12 cycles to finish the computations of all POPs by Eq. (10). More importantly, H engines of each PE share the same weights at each cycle which can reduce the bit-width of weights transmission to save power. Besides, for FC layer with $NON = 2$ and $NIN = 12$, we get $PON_m = 2$ and $PIN_m = 6$ by Eq. (10). As illustrated in Fig. 4 (b), the PE just needs 2 cycles to produce 2 POPs for the output neurons. Therefore, the computing arrays can

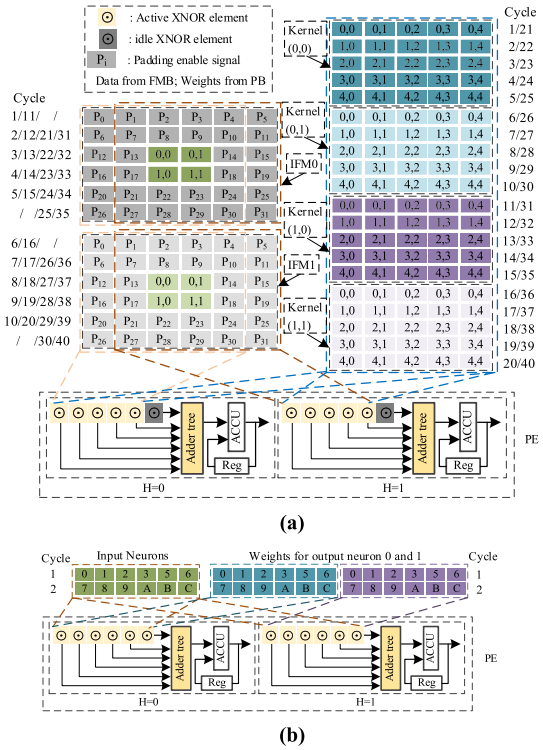

Fig. 4 Examples for Data allocation technology of FMAM and WAM. (a) $\mathbf{5} \times 5$ CONV operations; (b) **FC** operations.

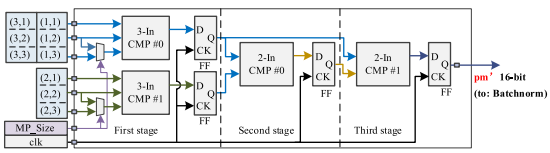

Fig. 5 Three-stage-comparison (TSCMP) engine.

efficiently support different sizes of padded CONV operations such as 3×3, 5×5, etc., and FC operations.

3.4 Pooling Design

The Pooling block receives the instruction from LLCR which has **MP_En** and **MP_Size** signals. When **MP_En** is low, the block is set to Bypass mode, meaning that POPs are forward directly to Batchnorm block. Otherwise, it is set to either 2×2 or 3×3 MP by **MP_Size**. To ensure the block processes these POPs in a pipeline fashion, it insists of $N \times (H/3)$ TSCMP engines, which can avoid using a great deal of storage to store POPs. As shown in Fig. 5, we employ Eq. (6) to design TSCMP. More importantly, to be compatible with 2×2 and 3×3 MPs, the TSCMP contains 3 row buffers of 3 16-bits each shown in the bule boxes of Fig. 5. Initially, it receives POPs then stores them into the row buffers. Afterwards, for 3×3 MP, while 3 rows and 3 columns of POPs are ready, the TSCMP also takes the max value after three stages; For 2×2 MP, the TSCMP stops reading new POPs at the third stage, and $\{(1,3),\ (2,3),\ (3,3)\}$ are always 0, where (i, j) represent the i^{th} row and j^{th} column.

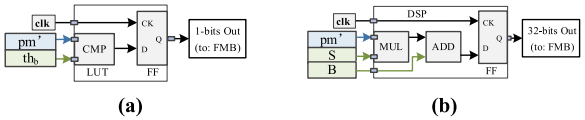

Fig. 6 (a) Comparison engine; (b) linear-operation engine.

Hence, while 2 rows and 2 columns of POPs are ready, the TSCMP also takes the max value after three stages. Note that both for the two MP operations, the oldest row buffer is refilled with the next row of POPs at the third stage since the oldest row of POPs will never be used. Consequently, the result pm' is send to the Batchnorm block.

3.5 Batchnorm Design

The Batchnorm block receives **BN_Type** signal from LLCR. When **BN_Type** is high/low, the Batchnorm is set to **BN-BA/BN-NBA** when the activation type is binary/non-binary. As shown in Fig. 6 (a) and (b), Eqs. (8) and (9) are employed to design the two types of BN engines, Comparison engine and Linear-operation engine. Initially, the block receives the result from MP pm' and starts computing. For Comparison engine, it produces 1 if $pm' \geq th_{l,n}$, otherwise, it produces 0; For Linear-operation engine, its output is equal to $pm' * A + B$. Consequently, the output activation stored in FMB is used to the input of the next macro layer. Note that $th_{l,n}$, A and B are done offline without additional computing cost in hardware inference.

3.6 On-Chip Memory (OCM) Deign

OCM, including FMB and PB, receives the instruction and control from LLCR and Global Controller to complete data transmission. To overlap transmission with computing, FMB employs the popular ping-pong technique. Hence, it has two buffers, which both can provide IFMs and save OFMs. After each layer done, the two buffers switch their functions each other.

To improve the efficiency of data read and write, the storage strategies of FMB and PB are needed to cooperate with FMAM and WAM, respectively. As introduced in Sect. 3.3, suppose the CONV operation needs inputs of the nr^{th} row in all the PIC_m input channels of the t_i tile, $PIC_m * NC$ inputs are retrieved from $t_i * nr$ address of FMB and reserved the corresponding $PIC_m * PICOL_m$ inputs by FMAM. Meanwhile, $POC_m * PIC_m * K_l$ weights are retrieved from $WAddr_{CONV}$ address of PB shown in Eq. (11); Additionally, suppose the FC operation needs inputs of the $TIN = NIN/PIN_m$ tile, PIN_m inputs and $PON_m * PIN_m$ weights are retrieved from TIN address of FMB and $WAddr_{FC}$ address of PB shown in Eq. (11), respectively. Afterwards, the inputs and weights are fed into the corresponding PEs by FMAM and WAM. Consequently, the outputs are stored into FMB for the next layer by FMAM.

$$WAddr_{CONV} = \frac{noc}{POC_m} * \frac{nic}{PIC_m} * K_l + \frac{nic}{PIC_m} * K_h + K_l$$

$$WAddr_{FC} = \frac{noe}{PON_m} * \frac{nie}{PIN_m} + \frac{nie}{PIN_m} \tag{11}$$

Where nic and noc are the nic^{th} input and noc^{th} output channels in the CONV layer; nie and noe are the nie^{th} input and noe^{th} output neurons in the FC layer.

4. Analytical Model

To simultaneously achieve high performance and high flexibility, an analytical model is introduced to determine the optimal design parameters of the computing array including N, H and W.

Firstly, build a LUT resource cost model. By using Xilinx Vivado design suite, we observe the resource cost of the three basic and most operation units including the XNOR element, NOT element, and 2-input unsigned and signed fixed-point adder, as shown in Table 2. Hence, the LUT_{CA}, the cost of the computing array, can be estimated as:

$$\begin{aligned}
LUT_1 &= \underbrace{N_1 * H_1 * W_1 * 1}_{LUTs\ for\ XNOR} + \underbrace{N_1 * H_1 * \left(\sum_{i=1}^{I}\left(\frac{W_1}{2^i} * i\right)\right)}_{LUTs\ for\ Adder\ trees\ of\ BCA} \\
&\quad + \underbrace{N_1 * H_1 * I}_{LUTs\ for\ ACCU} \\
I &= \lceil \log_2 W_1 \rceil \\
LUT_2 &= \underbrace{(N_2 * H_2 * W_2) * 4}_{LUTs\ for\ NOT} + \underbrace{N_2 * H_2 * \left(\sum_{j=1}^{J}\left(\left(\frac{W_2}{2^j}\right) * (8+j)\right)\right)}_{LUTs\ for\ Adder\ trees\ of\ NBCA} \\
&\quad + \underbrace{N_2 * H_2 * (8+J)}_{LUTs\ for\ ACCU} \\
J &= \lceil \log_2 W_2 \rceil \\
LUT_{CA} &= LUT_1 + LUT_2 < 50\% * LUT_{FPGA}
\end{aligned} \tag{12}$$

Where (N_1, H_1, W_1) and (N_2, H_2, W_2) are the scale parameters of BCAs and NBCAs which employ LUT_1 and LUT_2 numbers of LUTs, respectively. I and J represent the numbers of levels in adder trees of BCAs and NBCAs, respectively. The i^{th} and j^{th} levels have $(W_1/2^i)$ and $(W_2/2^j)$ numbers of 2-input adders, and their bit-widths are i-bits and $(7 + j)$-bits, respectively. Therefore, each adder in the i^{th} and the j^{th} levels adder trees uses i and $(8 + j)$ LUTs, respectively. Furthermore, LUT_{CA}, the sum of LUT_1 and LUT_2, is smaller than half of LUT_{FPGA} which is the available number of LUTs in the target FPGA. The ratio of 50% is got by engineering experience, as presented in Sect. 6.1.

Secondly, build a compatibility model. We observe that the structures of most BNNs have the following characters. Firstly, the input channel and neuron dimensions in most CONV and FC layers are multiples of 16; Secondly, the minimum width of FM (WFM_{min}) is generally equal to 8 in CONV layers. Finally, the popular kernel sizes are 3×3, 5×5, 7×7, and 11×11, and the sizes of popular MPs are 2×2 and 3×3. Furthermore, combining the analysis in Sect. 3.3, N is equal to multiples of 16 for the computations of output channels; H is equal to multiples of 3 due to 3×3 MP, and it is equal to or greater than WFM_{min}; W is equal to or greater than multiples of the production 16 times K_l, since it needs to satisfy the CCTP pattern. Moreover, we give priority to the computing efficiency of $K_l = 3$ which is the most popular kernel size. Therefore, the constraint conditions can be expressed as follows:

$$\begin{aligned}
&N = 16 \times n_1 \\
&H = 3 \times n_2 \geq WFM_{min} \\
&W \geq K_l \times 16 \times n_3,\ (K_l = 3)
\end{aligned} \tag{13}$$

Here, n_1, n_2 and n_3 are positive integers. The value of n_3 is aimed at making the number of the idle computing elements as small as possible when the computing arrays perform CONV operations of each popular kernel size.

Thirdly, build a performance model. As shown in Eq. (10), we can get the relationship between the number of clock cycles and the scale parameters for CONV and FC layers shown as follows:

$$\begin{aligned}
T_{CONV} &= \left\lceil \frac{NIC * K_l * K_l}{W} \right\rceil * \left\lceil \frac{NOC}{N} \right\rceil * \left\lceil \frac{NC}{H} \right\rceil * NR \\
T_{FC} &= \left\lceil \frac{NIN}{W} \right\rceil * \left\lceil \frac{NON}{H * N} \right\rceil
\end{aligned} \tag{14}$$

Finally, we design a python parsing program to explore the scale parameters automatically, which has three stages:

1) Get the respective ranges of (N_1, H_1, W_1) and (N_2, H_2, W_2) by Eq. (13).
2) Iterate over every possibility in their ranges, and then choose automatically all sets which can achieve the higher and similar theoretical performance by Eq. (14).
3) Choose the best set, which consumes the least resource cost by Eq. (12), among all the sets which are got in stage 2.

After finishing the three stages, we get (N_1, H_1, W_1) and (N_2, H_2, W_2) are $(16, 9, 96)$ and $(16, 9, 16)$, respectively.

Table 2 Resource cost of basic operations.

Operation	Precision (bit)	Slice LUT
XNOR element	Input: 1; weight:1	1
NOT element	Input: 8; weight:1	4
Unsigned fixed-point adder of BCA	Input: x	x
Signed fixed-point adder of NBCA	Input: x	x+1

5. Mapping Flow

To efficiently support change of the target BNN and acceleration of a new BNN, a mapping flow for FCA-BNN is proposed, as shown in Fig. 7. This flow includes five steps as follows:

Step 1: Generating of bitstream. We employ Verilog-HDL language to design FCA-BNN with the hardware-oriented optimization techniques and the optimal design parameters, and then use Vivado to generate the bitstream.

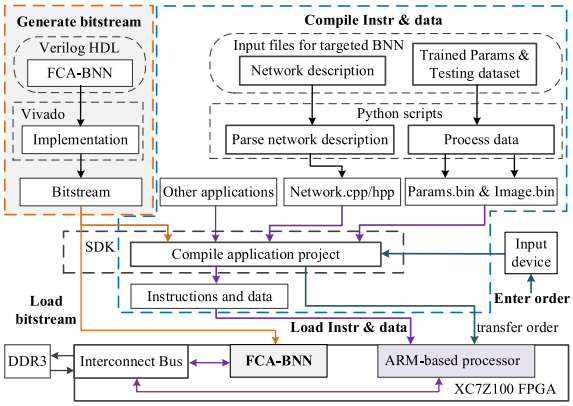

Fig. 7 Overview of the mapping flow.

Step 2: loading of bitstream. We use Vivado SDK load the bitstream into the target Xilinx XC7Z100 FPGA.

Step 3: compiling of instructions and data. We employ the loop tiling technique for different sizes of padded CONV and FC operations to design the script for parsing network description, which can parse the target network layer by layer to produce two C language files, network.cpp and network.hpp. These two files insist of the instructions of LLCR. Additionally, the script for processing data is developed by the storage strategies of FMB and PB, and processes the trained parameters and testing dataset to produce binarized parameters file (parameter.bin) and 8-bits fixed-point input file (images.bin), respectively. Afterwards, using SDK compiles the application project, including the produced four files and other applications applied to all the target networks, to generate the instructions and data.

Step 4: Loading of instructions and data. Under the communication between PC and ARM-based Processor, using SDK loads the instructions and data into LLCR and DDR3, respectively. Afterwards, FCA-BNN can accelerate the target network successfully.

Step 5: Entering of order. Each target BNN has its own order after executing the above four steps. Therefore, we just need to enter the corresponding order while changing the target BNN among the existing BNNs.

Even for a new network, we just perform the step 3 and 4 to map it into FCA-BNN, making the accelerator designer accelerate the new network quickly. Note that one of the five types of macro layers shown in Table 1 can match with each layer of the new BNN. In short, our mapping flow completes switch of the target network and acceleration of a new network quickly, without generating and loading bitstream.

6. Experimental Results

In this section, hardware implementation and preparation of the target BNNs are first introduced. Afterwards, we provide evaluation and comparison with previous designs. Finally, for change of the target BNN and acceleration of a new target BNN, comparison with the existing relevant works is given. The LUT resource utilization and power consumption are reported in Vivado after implementation.

Table 3 Resources utilization of each block.

	LUTs	BRAM	DSP
CA-BNN	76.1K	0	10
OCM	13.6K	692	0
LLCR and Global Controller	58.6K	0	0
Interconnect Bus	4.5K	0	0
Total	152.8K	692	10

Table 4 Structures of the models used to evaluate FCA-BNN. $CONV_{i,o,k,s}$ denotes a CONV layer with i input channels, o output channels, kernel size k × k and stride s. $FC_{i,o}$ denotes a FC layer with i input neurons and o output neurons. $MP_{k,s}$ is a max-pooling layer with window size k × k and stride s. BN-BA and BN-NBA have introduced in Sect. 3.5.

Model	Network architecture
LFC	$FC_{784,1024}$, BN-BA, $FC_{1024,1024}$, BN-BA, $FC_{1024,1024}$, BN-BA, $FC_{1024,10}$, BN-NBA
VGG-like	$CONV_{3,128,3,1}$, BN-BA, $CONV_{128,128,3,1}$, $MP_{2,2}$, BN-BA, $CONV_{128,128,3,1}$, BN-BA, $CONV_{128,256,3,1}$, $MP_{2,2}$, BN-BA, $CONV_{256,512,3,1}$, BN-BA, $CONV_{512,512,3,1}$, $MP_{2,2}$, BN-BA, $FC_{8192,1024}$, BN-BA, $FC_{1024,1024}$, BN-BA, $FC_{1024,10}$, BN-NBA
Cifar-10 AlexNet	$CONV_{3,96,5,1}$, $MP_{3,2}$, BN-BA, $CONV_{96,256,5,1}$, $MP_{3,2}$, BN-BA, $CONV_{256,384,3,1}$, BN-BA, $CONV_{384,384,3,1}$, BN-BA, $CONV_{384,256,3,1}$, $MP_{3,2}$, BN-BA, $FC_{4096,1024}$, BN-BA, $FC_{1024,1024}$, BN-BA, $FC_{1024,10}$, BN-NBA

6.1 Hardware Implementation

Table 3 shows the details of resources utilization for each block. FCA-BNN consumes about 152.8K LUTs, 692 36K BRAMs and 10 DSPs. For the demand of LUTs, CA-BNN accounts for ∼50% of used LUTs which is nearly equal to the estimate by Eq. (12), while the other half is mainly used to the designs of LLCR and Global Controller. Moreover, 10 DSPs are used for the linear-operations of BN-NBA. For requirement of storage, all the parameters of most models can be stored in OCM, while for large models, a tile-based parameter storage strategy is introduced, which takes only parameters required for the current tile from DDR3.

6.2 BNN Models Preparation

To fully evaluate FCA-BNN, we prepare three well-known BNNs with different structures including LFC [16] on MNIST, VGG-like [6] on Cifar-10 and Cifar-10 AlexNet which is inspired from AlexNet [21], as shown in Table 4.

6.3 Functionality Evaluation and Comparison

The inference of three networks on GPU and FPGA (FCA-BNN) platforms both classify 10,000 test images to get the $Accu_{GPU}$ and $Accu_{FPGA}$ accuracies, respectively, and then the ***DoIA*** is equal to $Accu_{GPU}$ minus $Accu_{FPGA}$. Therefore, the smaller ***DoIA*** represents network inference on FCA-BNN is more reliable. As shown in Table 5, for LFC and MLP which are four-layer FC models, our LFC achieves ***DoIA*** of 0.07%, which is less than that of the works [10], [16]. For VGG-like, the work [16] just has accuracy of 80.1%. Although [12] achieves 0.17% smaller ***DoIA***, it uses the odd-even padding to makes control logic

Table 5 Evaluation of accuracy: LFC, VGG-like and AlexNet for Cifar-10.

	[10]	[16]	Ours	[12]	[16]	[17]	Ours	[17]	Ours
Model	MLP	LFC	LFC	VGG-like				AlexNet	Cifar-10 AlexNet
$Accu_{GP}$ (%)	99.3	98.81	98.86	88.75	-	89.06	86.11	79.4	83.14
$Accu_{FPGA}$ (%)	97.7	98.4	98.79	88.61	80.1	86.31	85.80	66.8	82.74
DoIA (%)	1.6	0.41	0.07	0.14	-	2.75	0.31	12.6	0.40

Table 6 Evaluation of performance: LFC and VGG-like.

	[10]	[16]	Ours	[7]	[12]	[16]	Ours
Model	MLP	LFC	LFC	VGG-like			
Clock (MHz)	200	200	166	200	-	200	166
KLUTs	-	82.9	152.8	180.9	29.6	46.3	152.8
Power (W)	12.9	22.6	5.7	9.0	3.3	11.7	5.7
Latency (us)	-	2.44	3.0	609	1,920	501	371
GOPS	7,373	9,086	1,932	2,026	722	2,465	3,322
GOPS/W	424	402	339	225	219	211	583

Table 7 Cross-platform evaluation of performance: AlexNet for Cifar-10.

Platform	Intel i7-6700	NVIDIA GTX1060	XC7Z100 FPGA
Model	Cifar-10 AlexNet		
Clock (MHz)	2.6K	1.5K	166
Power (W)	65	120	5.7
Latency (us)	4,099	702	245
GOPS/W	2.8	8.7	527.0

more complex. For Cifar-10 AlexNet, our ***DoIA*** of 0.4% is smaller than that of [17], since this work adopts linear shift operations to approximate multiplications in BN operations and ignores bias of CONV and FC layers. In short, FCA-BNN employs 0-padding and threshold-based comparison including bias to achieve small ***DoIA***.

6.4 Performance Evaluation and Comparison

Comparing with the previous state-of-the-art works, FCA-BNN achieves competitive performance on LFC and VGG-like models, as shown in Table 6. For LFC, despite our throughput of 1,932 *GOPS* is 4.7/3.8× lower than that of [10] and [16], our latency of 3*us* is nearly equal to theirs and our power consumption of 5.7*W* achieves at least 2× lower than theirs. Moreover, our energy-efficiency of 339 *GPOS/W* is 0.8× by theirs. But for VGG-like, FCA-BNN achieves latency of 371*us*, throughput of 3,322 *GOPS* and energy efficiency of 583 *GPOS/W*, which are at least 1.4/1.3/2.6× better than the designs [7], [12] and [16]. Despite our power consumption is 1.7× higher than that of the design [12], our energy-efficiency is 2.7× better than that of the design [12].

To the best of our knowledge, Cifar-10 AlexNet is the first work to accelerate binarized AlexNet model on Cifar-10, and thus the performance results of this model on FCA-BNN are compared with that of Intel i7-6700 (CPU) and NVIDIA GTX1060 (GPU), as shown in Table 7. FCA-BNN achieves at least 16.7/2.9× smaller in latency over CPU/GPU. Moreover, FCA-BNN achieves 188.2/60.6× better in energy efficiency over CPU/GPU.

As we have seen, FCA-BNN on VGG-like achieves the better performance results than the results on LFC and Cifar-10 AlexNet. Since LFC is just composed of four layers, the overall results are lager limited by the first layer. For Cifar-10 AlexNet, the kernel sizes of the first two layers are 5×5, while we give priority to the computing efficiency of 3×3 kernel shown in Eq. (13). But for Cifar-10 AlexNet, it is more suitable than VGG-like in this kind of scenario where low-latency is highly desired.

Table 8 Evaluation of efficiency: the mapping flow.

Step	[16]	[17]	[19]	Ours	[16]	[17]	[19]	Ours
	change of the target BNN				acceleration of a new BNN			
loading bitstream	Y	Y	Y	N	Y	Y	Y	N
generating bitstream	N	N	N	N	Y	Y	Y	N
Compiling instructions	N	N	N	N	Y	Y	Y	Y
loading instructions	Y	Y	Y	N	Y	Y	Y	Y
Enter order	N	N	N	Y	N	N	N	N

6.5 Evaluation and Comparison of Mapping Flow Efficiency

Table 8 shows the steps required to perform change of the target BNN and acceleration of a new BNN on FCA-BNN and the prior works [16], [17], [19]. As presented in Sect. 5, the mapping flow for FCA-BNN just needs the step of entering order which generally takes less than one second to change the target BNN, while the existing designs require the two steps, loading bitstream and instructions, which generally take tens of seconds for operating and executing. More importantly, the next target BNN is switched seamlessly with the current running BNN on FCA-BNN.

Besides, for accelerating a new BNN, we only perform compiling and loading instructions, while other works also need the extra two steps, loading and generating bitstream, which generally take server hours for the large-scale design. Note that there is a requirement of FCA-BNN: each macro layer of the new BNN can match with one of the provided five types of macro layers shown in Table 1. Nevertheless, these five types of macro layers can support the most popular BNNs. Moreover, FPGA can be reprogrammed to respond to new advances of BNNs, making FPGA-based FCA-BNN more applicable in such a fast-changing field than ASICs. Hence, FCA-BNN is applicable and promising in mobile devices.

7. Conclusion

In this paper, we propose a flexible and configurable accelerator. FCA-BNN employs the layer-level configurable technique (LLCR) to execute each layer of target BNN in a pipeline fashion. Moreover, we use the hardware-oriented optimal formulas to design energy-efficient computing arrays for CONV and FC operations, TSCMP engine for

MP operations, and the two types of BN engines including bias for BN operations. Besides, the analytical model is used to determine the scale parameters, achieving high flexibility and performance. The inference accuracies of LFC, VGG-like and Cifar-10 AlexNet on FCA-BNN are just 0.07/0.31/0.4% less than that of the three BNNs on GPU, respectively. In term of energy-efficiency, FCA-BNN achieves 0.8× for LFC and 2.6× for VGG-like compared with the existing works. Furthermore, for Cifar-10 AlexNet, FCA-BNN achieves 188.2/60.6× energy-efficiency better than CPU and GPU, respectively. To the best of our knowledge, by using our proposed mapping flow, FCA-BNN is the most efficient in change of the target BNN and acceleration of a new BNN compared with previous works. Meanwhile, FCA-BNN keeps the competitive performance.

Acknowledgements

This work was supported in part by the National Natural Science Foundation of China under Grant No. U20A20202.

References

[1] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-eye: A complete design flow for mapping cnn onto embedded fpga," IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol.37, no.1, pp.35–47, 2018.

[2] L. Lu, J. Xie, R. Huang, J. Zhang, W. Lin, and Y. Liang, "An efficient hardware accelerator for sparse convolutional neural networks on fpgas," IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM), San Diego, CA, USA, pp.17–25, 2019.

[3] L. Gong, C. Wang, X. Li, and X. Zhou, "WiderFrame: An automatic customization framework for building cnn accelerators on fpgas: Work-in-progress," International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), Shanghai, China, pp.5–7, 2020.

[4] A. Sohrabizadeh, J. Wang, and J. Cong, "End-to-end optimization of deep learning applications," ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), pp.133–139, 2020.

[5] C. Zhu, K. Huang, S. Yang, Z. Zhu, H. Zhang, and H. Shen, "An efficient hardware accelerator for structured sparse convolutional neural networks on fpgas," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.28, no.9, pp.1953–1965, 2020.

[6] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: training deep neural networks with weights and activations constrained to +1 or −1," arXiv preprint arXiv:1602.02830, 2016.

[7] T. Geng, A. Li, T. Wang, C. Wu, Y. Li, R. Shi, W. Wu, and M. Herbordt, "O3BNN-R: An out-of-order architecture for high-performance and regularized bnn inference," IEEE Trans. Parallel Distrib. Syst., vol.32, no.1, pp.199–213, 2021.

[8] T. Geng, T. Wang, C. Wu, C. Yang, S.L. Song, A. Li, and M. Herbordt, "LP-BNN: Ultra-low-latency bnn inference with layer parallelism," IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP), New York, NY, USA, pp.9–16, 2019.

[9] J. Wang, X. Jin, and W. Wu, "TB-DNN: A thin binarized deep neural network with high accuracy," 22nd International Conference on Advanced Communication Technology (ICACT), Phoenix Park, Korea (South), pp.419–424, 2020.

[10] S. Liang, Y. Lin, W. He, L. Zhang, M. Wu, and X. Zhou, "An energy-efficient bagged binary neural network accelerator," IEEE 3rd International Conference on Electronics Technology (ICET), Chengdu, China, pp.174–179, 2020.

[11] R. Zhao, W. Song, W. Zhang, T. Xing, J.-H. Lin, M. Srivastava, R. Gupta, and Z. Zhang, "Accelerating binarized convolutional neural networks with software-programmable fpgas," International Symposium on Field-Programmable Gate Arrays (FPGA), ACM, pp.15–24, 2017.

[12] P. Guo, H. Ma, R. Chen, P. Li, S. Xie, and D. Wang, "FBNA: A fully binarized neural network accelerator," IEEE International Conference on Field Programmable Logic and Applications (FPL), pp.51–54, 2018.

[13] N. Fasfous, M.R. Vemparala, A. Frickenstein, and W. Stechele, "OrthrusPE: Runtime reconfigurable processing elements for binary neural networks," Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, France, pp.1662–1667, 2020.

[14] X. Lin, et al., "Towards accurate binary convolutional neural network," International Conference on Neural Information Processing Systems (NIPS), pp.344–352, 2017.

[15] D. Wang, K. Xu, J. Guo, and S. Ghiasi, "DSP-efficient hardware acceleration of convolutional neural network inference on fpgas," IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol.39, no.12, pp.4867–4880, 2020.

[16] Y. Umuroglu, N.J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), ACM, pp.65–74, 2017.

[17] S. Liang, S. Yin, L. Liu, W. Luk, and S. Wei, "FP-BNN: binarized neural network on fpga," ELSEVIER Neurocomputing, pp.1072–1086, 2017.

[18] M. Ghasemzadeh, M. Samragh, and F. Koushanfar, "ReBNet: Residual binarized neural network," IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp.57–64, 2018.

[19] E. Wang, J.J. Davis, P.Y.K. Cheung, and G.A. Constantinides, "LUTNet: Learning fpga configurations for highly efficient neural network inference," IEEE Trans. Comput., vol.69, no.12, pp.1795–1808, 2020.

[20] S. Muralidharan, K. O'Brien, and C. Lalanne, "A semi-automated tool flow for roofline analysis of opencl kernels on accelerators," Proc. 1st Intl. Workshop on Heterogeneous High-performance Reconfigurable Computing (H2RC), Austin, TX, USA, 2015.

[21] A. Krizhevsky, I. Sutskever, and G.E. Hinton, "ImageNet classification with deep convolutional neural networks," International Conference on Neural Information Processing Systems (NIPS), pp.1097–1105, 2012.

[22] M.D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," European Conference on Computer Vision (ECCV), 2014.

Jiabao Gao received the M.S. from South China University of Technology, Guangzhou, China, in 2013. He is currently working toward the Ph.D. degree at the School of Microelectronics, Fudan University, Shanghai, China. His current research interest is hardware accelerators for neural network.

Yuchen Yao received the B.E. and M.S. degrees from East China Normal University and Fudan University in 2017 and 2020, respectively. Her research interest is FPGA acceleration of BNN.

Zhengjie Li received the B.S. and M.S. degrees from UESTC in 2006 and 2012, respectively. During 2006-2018, he worked in CSMT as an FPGA chip designer. He now is a PhD student in Fudan University, his research interest is FPGA acceleration of DNN.

Jinmei Lai received the Ph.D. degree in School of Electricity from Shanghai Jiao Tong University, Shanghai, China, in 1998. In 2003, she joined Fudan University, Shanghai, China, where she is currently a Full Professor at the State Key Laboratory of ASIC and System, School of Microelectronics. Her current research interests include design of deep learning R&D platform based on SoC FPGA, and embedded programmable IP core and its EDA software algorithm.