A High-throughput Scalable BNN Accelerator with Fully Pipelined Architecture

Zhe Han, Jingfei Jiang*, Jinwei Xu, Peng Zhang, Xiaoqiang Zhao, Dong Wen, Yong Dou
Science and Technology on Parallel and Distributed Processing Laboratory
National University of Defense Technology
Changsha, China
{hanzhe18, *jingfeijiang, xujinwei13, zhangpeng14f, zhaoxiaoqiang18, yongdou}@nudt.edu.cn, 310-we-aaa-1@163.com

***Abstract*—By replacing multiplication with bit operation, Binarized Neural Networks (BNN) are extremely suitable for FPGA acceleration. Previous researches highlighted the potential exploitation of BNNs performance. However, present research approaches to improving chip efficiency have not fully exploited the potential of BNNs. Thus, we proposed a scalable fully pipelined BNN architecture, which targeted on maximizing throughput. By exploiting multi-levels parallelism and balancing pipeline stages, it achieved excellent performance. Moreover, we shared on-chip memory and balanced the computation resources to further utilizing resource. Then a methodology is proposed that explores design space for the optimal configuration. This work is evaluated based on Xilinx UltraScale XCKU115. The results show that the proposed architecture achieves 2.24x-11.24x performance and 2.43x-11.79x resource efficiency improvement compared with other BNN accelerators.**

***Index Terms*—Convolutional neural network (CNN), Binarized convolutional neural network (BNN), field-programmable gate array (FPGA)**

I. Introduction

Recently, Convolutional Neural Networks (CNNs) have gained great success and show dramatic performance in many tasks [1] [2] [3]. However, heavy computations and intensive parameters significantly restrict their utilization on real-time or resource-constrained platforms. Recent researches [4] focus on reducing memory and computation cost have obtained certain achievement. The binarized convolutional neural network is one of them.

Many works [5] [6] have shown that floating-point numbers are unnecessary for inference. Based on quantization techniques, Courbariaux et al [7]. proposed first binarized method named BinaryConnect. They compressed full-precision values of weights into binary values which showed the effectiveness of binarization. Later, Courbariaux et al. further developed the binarized neural network and proposed BinaryNet [8]. Since then, many approaches [9] [10] [11] were proposed for improving the BNN accuracy especially in large dataset like ImageNet.

With the advantage of reconfigurable logic, FPGA is very suitable to accelerate BNN. FPGAs can exploit the benefits from bitwise operations while CPUs and GPUs are more capable in integer and floating-point tasks. Zhao et al. [12] first implemented an accelerator for binarized neural networks on FPGA. Their approach gets twice the performance of CPU in a small FPGA chip which only consumes 4.7 Watt energy. Considering the first layer of [12] is still in floating-point, Guo et al. [13] proposed a method to binarize the input before inference which further reduced the chip area. Yang et al. [14] replaced the original binary convolution layer with two parallel binary convolution layers and got a smaller weight which can be fully stored on-chip memory. All these works proved the potential of BNN on FPGA.

However, most of them focused on area optimization and implemented with tiny FPGA chips. They folded the neural networks and reused computation units, which cannot exploit the benefits of pipeline and on-chip memory completely. These architectures were only suited for strict-constrained resource designs where high throughput was not needed. In order to exploit the full computing power of large FPGAs, we proposed a scalable BNN accelerator with fully pipelined architecture which targeted on high throughput scenarios. Each stage in the architecture can be processed with different layer blocks, which provides a high throughput and efficiency. For higher performance, we analyzed operations of each layer and optimized design to get a balanced pipeline. By sharing on-chip memories and balancing computation resources, we got a higher resource utilization and efficiency. We used the proposed design space exploration algorithm to generate configurations and implement them on Xilinx UltraScale XCKU115 FPGA. The results demonstrate that our design improves performance at least twice as much as previous works. The main contributions of this work are as follows:

- We proposed a fully pipelined scalable BNN accelerator that exploits multi-levels of parallelism, including task level, layer level, loop level, and operation level.
- We proposed a strategy to optimize resource utilization.
- We proposed a design space exploration methodology to search for the optimal configurations under the FPGA constraints.
- We implemented six different configuration BNN [8] accelerators on a Xilinx XCKU115 board and achieves significant improvement compared with prior BNN accelerators.

II. Background

We focus on one of the most researched BNN form [8], which mainly consists of convolutional layers and fully connected layers. This section will briefly introduce these layers.

A. Convolutional Layer

The convolutional layer is to generate a set of feature maps by sliding kernels on the input feature maps and convolving. At each sliding position, a product of the corresponding elements and a sum operation are run between the kernel and the input image to project the information from the input feature maps to an element in the output feature map. Assuming X represents the input, Y represents the output, and K represents the kernel, the convolutional layer can be expressed as:

$$Y_i = \sum_{j=1}^{n} X_j \otimes K_{ij} + bias \qquad (i = 1, 2, ..., m) \qquad (1)$$

Assuming that the size of the kernel is $k \times k$, $k \times k$ multiply-accumulate operations are required for each sliding window. Therefore, the convolutional layer has a high computational complexity. In order to simplify the computation, we use the sign of data to binarize feature maps and the kernel, which means we can use 1 bit to represent the data(-1-¿0, +1-¿1). Therefore, we can use XNOR operation to replace the expensive multiplication equivalently. In particular, because the convolutional layer performs padding to ensure that the information on the edge of the input is retained, our binarized feature map actually has three values(+1, 0, and -1). Hence we use 2 bits for the representation of the input feature map at the same time 1 bit for kernels and output feature map.

B. Fully Connected Layer

The fully connected layer often acts as a "classifier" in the convolutional neural network. The core operation is the matrix-vector product. Assuming X represents the input image, Y represents the output result, and W represents the weight, the fully connected layer can be expressed as:

$$Y_i = \sum_{j=1}^{n} X_j * W_{ij} + bias \qquad (i = 1, 2, ..., m) \qquad (2)$$

Similar to the convolutional layer, we can use the XNOR operation to replace the multiplication since the input is binary. In addition, the fully connected layer does not need the padding operation. The computational data of the multiplication only needs 1 bit. Thus, we use the XNOR and Popcount instead of the multiplication and the addition tree to achieve fast multiplication and accumulation.

III. Hardware Architecture

This section describes the detailed implementation of our architecture.

A. Overall Architecture

Fig. 1 shows the hardware structure of our design. The accelerator communicates with the host computer through the PCI-Express (PCIE) bus. The images from the host computer and the results from the accelerator can be transmitted through DMA. FPGA chip mainly consists of controller, on-chip memories, input / output buffers and PEs. The controller is responsible for receiving instructions from the host computer and controlling the status of the accelerator. The on-chip memories are composed of different size BRAMs. They are used to store intermediate results and weights which saves bandwidth significantly. In order to improve resource utilization, the on-chip memories are shared by different PEs. The detail is described in Section IV-B. The input / output buffer consists of two 512 bits wide FIFOs, which are used to buffer data between off-chip DDR and on-chip memories. The BNN accelerator includes different numbers of processing elements (PEs) based on demand and resource constraints. Each PE output a result in a macro pipeline cycle. By assembling different numbers of PEs, we can achieve different parallelism at the task-level.

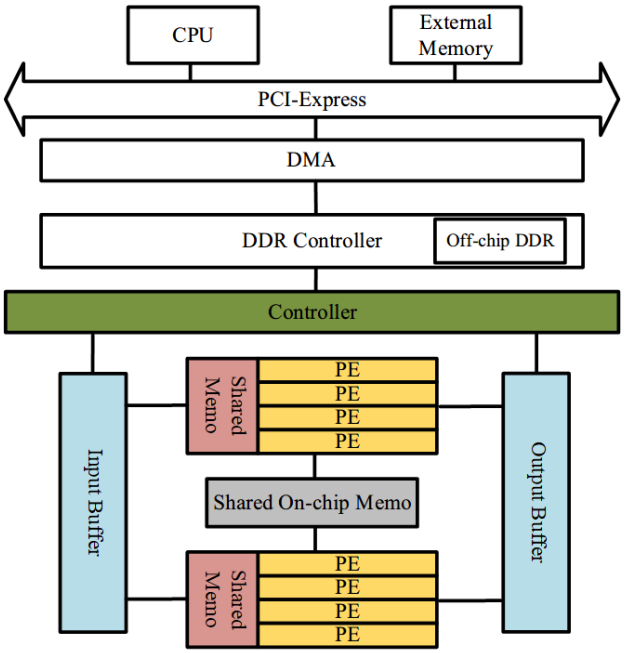

Fig. 1. Overall Structure of Our Accelerator

B. PE Design

As shown in Fig. 6, the PE consists of a series of layer block. There are two layer block types correspond to the convolutional layer and the fully connected layer. The memory block is used to connect the two layer block. It can regroup and buffer all intermediate feature maps. The PE is fully pipelined and each layer block is divided into different pipeline stages by memory blocks.

C. Layer Block Design

1) Convolutional Layer Block: As shown in Fig. 2, the convolutional layer block usually consists of four parts: mapping logic, convolutional units (CUs), batch normalization logic and pooling logic. The mapping logic generates a number of sliding windows as input for convolutional units. The convolutional unit is a small logic which performs multiple dot-products every pipeline cycle. Batch normalization

logic binarizes output of CUs by comparison with trained parameters. Pooling logic is optional. It performs max-pooling operations with OR gates.

The specific structure depends on loop-level parallelism which consists of four parts: input channel parallelism, output channel parallelism, convolutional window parallelism and operation parallelism. Of these four degrees of parallelism, the first three are used to design convolutional blocks. The different structures correspond to different combinations of these three that can meet different convolutional layer sizes and performance requirements. Fig. 2 shows the situation that the convolutional window parallelism is 32. The input feature map width is 16. Thus, the structure shown in the figure can output one row of feature maps per cycle.

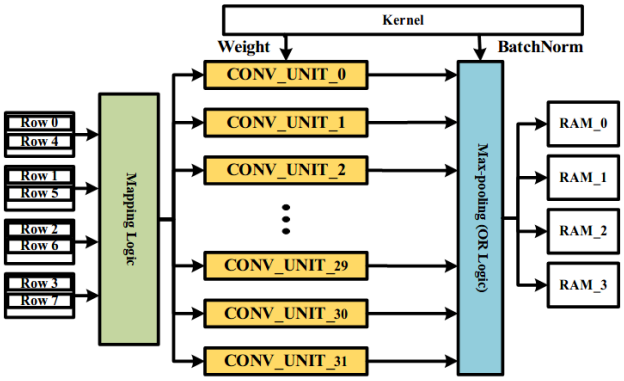

Fig. 2. Example of Convolutional Layer Block

2) Fully Connected Layer Block: The structure of the fully connected layer block is shown in Fig. 3. The fully connected layer block is composed of three parts: XNOR-based multiplier, popcount-based accumulation and batch normalization logic. Assuming the degree of parallelism is p, FC block read the feature map and weight of length p every cycle. Considering that the inputs and weights of the fully connected layer in the BNN are all in one bit, we use XNOR operation and popcount operation instead of vector multiplication and accumulation tree. Batch normalization logic binarizes output in the same way with the convolutional layer block.

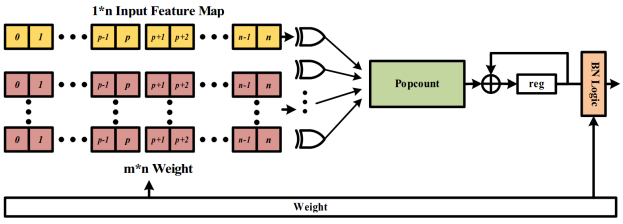

Fig. 3. The Architecture of Fully Connected Layer Block

D. Convolution unit

Assuming the convolution window size is $k \times k$, the operation parallelism is usually the same with the convolution window size. Therefore the convolution unit needs $k \times k$ input registers, $k \times k$ weight registers, $k \times k$ ternary multipliers, and an accumulation tree with $k \times k$ inputs as shown in Fig. 4. We use bit operation rather than multiplication which is similar to the fully connected layer block. However, the input of convolution unit has three values and is represented by two bits due to padding operations. We product input and weight by the ternary multiplier which consists of an AND gate and a XOR gate. In particular, we use addition instead of multiplication in the first convolutional layer where the input feature map is floating-point.

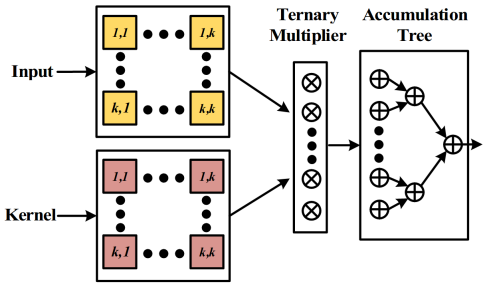

Fig. 4. The Architecture of Convolution Unit

E. Memory Block

The memory block consists of multiple banks of on-chip memory units (BRAMs). By separating BRAM, each bank has multiple data access interfaces and support simultaneous reading of K lines of data under the convolutional window. K is usually the same with the height of adjacent convolutional kernels. The size of a word is an integer multiple of output feature image width. For the memory block between two concatenated layer blocks, each bank will be double in order to send and receive at the same time. The specific configuration of the three parts is directly related to concatenated layer blocks' parallelism. The number of banks corresponds with input/output channel parallelism. The number of data access interfaces and bit-width depend on convolutional window parallelism. In Fig. 2, the input memory block has one bank of BRAM and the bank is divided into four parts for reading four lines of data simultaneous.

IV. Design Optimization

A. Pipeline Balancing

A good pipeline requires less dependencies and balanced pipeline stages. We fully expand the network structure and set up multiple memory blocks to divide each layer into a single stage which removes the data dependencies between different layers. However, there is a huge gap of workload between different layer blocks. The computational quantities for the BNN model are shown in Table 5. The quantities of computing operations of fully connected layer is very different from convolutional layers. Using the same degree of parallelism leads to a large amount of idle and wasted resources. Therefore, our strategy is to balance layer blocks'

execution time by setting different parallelism for them. The execution time can be estimated by (3).

$$execution\_time = \frac{operations}{parallelism_{loop}} \tag{3}$$

Fig. 5 shows the execution time of each layer block in our BNN accelerator. It can be seen that the execution time for hidden layers is perfectly balanced. The time of the first layer after adjusted is exactly half of the time of hidden layers, so we can balance its workload by using one first layer block for two PEs. The last layer has negligible waste because of its tiny workload (less than 1%).

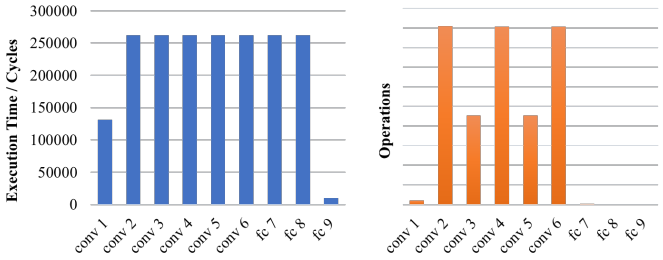

Fig. 5. Balanced Execution Time (left) and Operations (right) of Pipeline Stages

B. On-Chip Memory Sharing

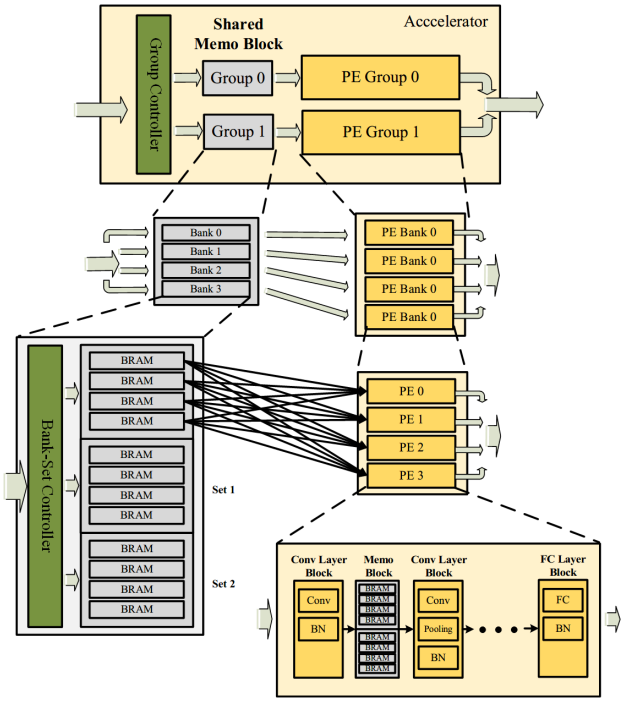

Fig. 6. Example of Memory Sharing. With the memory sharing, PEs are divided into corresponding groups.

It's obvious that the size of intermediate data from different layer block is different. Although FPGAs can implement operations and storage of any word size theoretically. In fact, the smallest unit of memory resources is not 1 bit. Taking

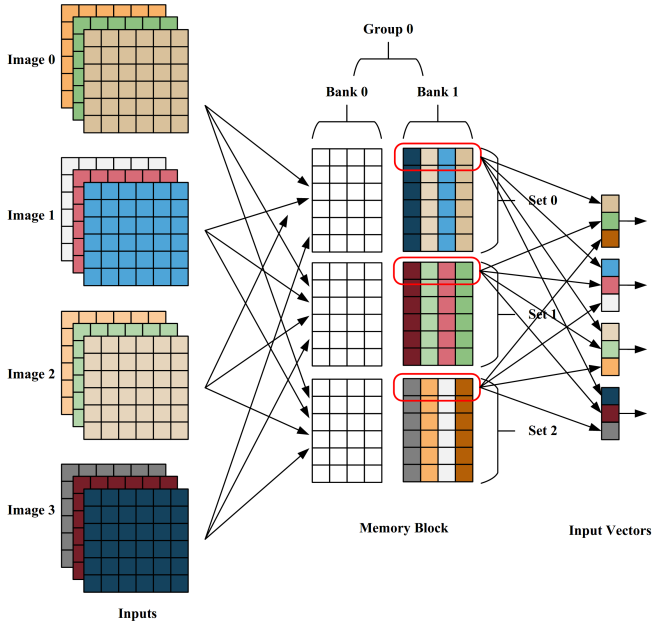

Fig. 7. Example of Loading from Shared Memory

Xilinx FPGA as an example, the smallest unit of on-chip memory (BRAM) is an 18K BRAM block with word size of 32 bits and depth of 512. That means wasted resources if our word size and data depth are not an integer multiple of 32 and 512. In addition, different degrees of parallelism also have certain requirements for data access as shown in Section III-E. The data must be stored in different on-chip memory blocks separately. For example, the input of each PE is a RGB image size of 32×32. As mentioned in Section III-E and Section IV-A, we set the parallelism of input channel and operation both to 3. There are three sets of BRAM in the input memory block and each set have 4 BRAM blocks. In this instance, the memory block consumes twelve 18K BRAM blocks for a 24Kb image according to the BRAM combination rules of Xilinx FPGA. Only 12.5% of memory are actually used. This is unacceptable for expensive on-chip memory resources. Therefore, we propose a strategy for sharing on-chip memory across PEs. By binding a certain number of PEs, multiple PEs can jointly use a set of BRAM. Fig. 7 and Fig. 6 show a example of the share and group strategy. For first memory block, our strategy only uses six 18K BRAM blocks and satisfies four PEs data requirements. The utilization reaches 100%.

C. Resources Balancing

The complex multipliers are replaced by ternary multipliers in BNN. Therefore, most DSPs are idle while LUT is facing a bottleneck due to insufficient resources. For higher task-level parallelism, we use the idle DSP to replace complex operations in the convolutional unit. Instead of multiplication, we use the DSP to replace the addition in the convolutional unit. This is because multiplication is only a small part of the operation after using ternary multipliers. Experiments show that we can

replace the accumulation tree in the convolutional unit with one DSP48 slice equivalently and reduce the LUT utilization of the convolution unit by more than half as shown in the Table I.

TABLE I
RESOURCE UTILIZATION OF DIFFERENT CONVOLUTIONAL UNIT

Strategy	LUT	FF	DSP
Unbalanced	37	25	0
Balanced	13	15	1

V. DESIGN SPACE EXPLORATION

As the above-mentioned design have a large range of parameters, it is difficult to find efficient configurations quickly. Therefore, we proposed a design space exploration algorithm to quickly search for the optimal configuration with constraints of throughput and resource.

A. Performance Analysis

1) Execution time: For a certain layer, the number of operations OP_{layer} is a constant. Therefore the execution time of the layer can be computed as follows:

$$t_{layer} = \frac{OP_{layer}}{p_{loop} \times p_{op}} \times \frac{1}{f} \tag{4}$$

Note that f is clock frequency. p_{loop} indicates loop-level parallelism and p_{op} indicates operation-level parallelism. p_{loop} contains three parallel types: input channel parallelism p_{ic}, output channel parallelism p_{oc} and convolutional window parallelism p_w. Their relationship is as Equation 5.

$$p_{loop} = p_{ic} \times p_{oc} \times p_w \times p_{op} \tag{5}$$

Once the pipeline been established, each pipeline cycle will return a result. For a pipeline with k stages, the establishment of the layer pipeline takes $k-1$ cycles. Thus, it takes $n+k-1$ cycles to calculate the results of n images. The length of the layer pipeline cycle is equal to t_{layer_max}, the maximum execution time among all layers. The execution time T is

$$T = t_{layer\_max} \times (n + k - 1) \tag{6}$$

2) Resource: Since we store all the weights and the intermediate feature maps with on-chip memories, we remove the bottleneck of off-chip memory bandwidth. Our restrictions mainly come from DSP, LUT and on-chip memory (BRAM). Assuming $\#Memory$ is the memory usage and $\#Memory_{PE}$ is the memory usage of one PE, therefore

$$\#Memory = \#Memory_{weight} + \#Memory_{PE} \times p_{task} \tag{7}$$

3) Throughput: The throughput is evaluated by GOP/s, which indicates the quantity of operations the accelerator performs in a second. The throughput of the accelerator can be computed as follows:

$$Throughput = \frac{p_{task} \times OP_{model}}{t_{layer\_max}} \tag{8}$$

B. Methodology

The target of design space exploration is to find the best configuration that maximizes the throughput while satisfies the restrictions of on-chip memory capacity and computation resources. Therefore the searching problem can be defined as follows

$$\begin{cases} \max(p_{task}) \\ \min(t_{layer\_max}) \end{cases} \tag{9}$$

Considering the design variables are all integers, we proposed an algorithm to search the optimal configuration. The pseudocode is shown followed.

for all p_{loop_i} **do**
  $p_{loop\_i} \leftarrow 1$
end for
$Throughput_{max} \leftarrow 0$
repeat
  calculate all t_{layer_i}
  $j \leftarrow \text{argmax}(t_{layer\_i})$
  $p_{loop\_j} \leftarrow p_{loop\_j} + 1$
  find the best combination of $(p_{ic}, p_{oc}, p_w, p_{op})$ with minimum t_{layer} and $\#Memory_{layer}$
  calculate p_{task}
  calculate $Throughput$
  if $Throughput \geq Throughput_{max}$ **then**
    $Throughput_{max} \leftarrow Throughput$
    print current configuration
  end if
until $\#LUT \geq LUT_{max}$**or**$\#Memory \geq Memory_{max}$

VI. EXPERIMENT

A. Experimental Setup

We re-trained BinaryNet model on Cifar-10 dataset and the test accuracy is 87.31%. Cifar-10 is a typical image classification task dataset. It contains 50,000 training images and 10,000 test images in 10 categories. The images are stored in RGB image format. To evaluate our design, we implemented it on the Xilinx Kintex UltraSclae evaluation board with XCKU115 FPGA chip. Our accelerators are implemented with Verilog. All synthesis results are obtained from Xilinx Vivado 2018.3. In order to verify the flexibility of our design, we have implemented six configurations of accelerators that can be deployed on FPGAs of different sizes based on performance requirements and resource constraints. The INA3221 chip can monitor the voltage and current of the entire FPGA evaluation board. We use it to measure the power consumption.

VII. RESULT

Table II shows performance and resource utilization details of our accelerator using different number of PEs. By comparing the resource and power efficiency of different configurations, we find that both resource efficiency and power efficiency are steadily increasing with task-level parallelism. The main reason for this is that different configurations use the same peripheral framework (such as DDR), so when the task level parallelism is very low, the acceleration logic is much

TABLE II
SUMMARY OF DIFFERENT CONFIGURED IMPLEMENTATIONS

Configuration	kLUT	BRAM	DSP	Time (ms)	kFPS	Througput (GOPS)	Power (W)	FPS/W	FPS/kLUT	GOPS/W	GOPS/kLUT
1PE	41	581	13	1.75	0.57	658	9.20	62.20	14.03	71.48	16.13
2PE	53	602	23	0.87	1.14	1315	9.68	118.22	21.53	135.86	24.75
4PE	78	644	43	0.44	2.29	2630	10.80	211.93	29.37	243.55	33.75
8PE	127	742	83	0.22	4.58	5261	12.95	353.49	35.91	406.22	41.27
16PE	239	937	163	0.11	9.16	10521	17.70	517.25	38.30	594.42	44.01
32PE	428	1321	323	0.05	18.31	21042	26.40	693.58	42.75	797.06	49.13
16PE-Balanced	198	937	2211	0.11	9.16	10521	21.00	435.97	46.14	501.01	53.02

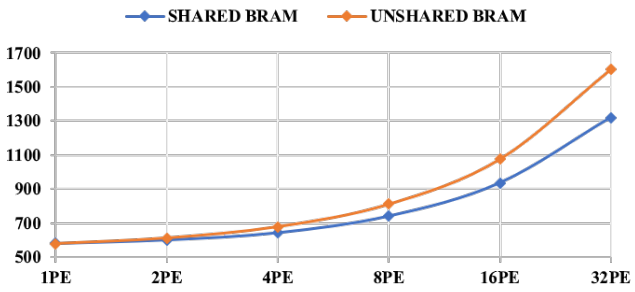

Fig. 8. Comparison of Shared Memory

smaller than the framework which consumes most resources. Therefore, as the parallelism increases, the advantages of our design will gradually become apparent. Table II also shows the result of 16 PEs implementation with balanced resources. The result indicates that we reached 46.14 FPS/kLUT, which exceeds the 42.75 FPS/kLUT of 32 PEs implementation.

Fig. 8 shows the resource utilization in both cases with and without memory sharing. It can be seen that as the parallelism increases, more on-chip storage will be used without storage sharing. With a PE of 32, the use of storage sharing can save about 565 18K BRAMs.

A. Comparison

As our design is scalable and targeted to high throughput, we implement prior works to similar LUTs utilization for fair comparison. Table III compares our implementations with prior works that have similar BNN model.

TABLE III
COMPARISON AGAINST PRIOR WORK

	Zhao et al. [12]	FBNA [13]	our-32PE
Clock (MHz)	143	143	150
Precision	1-2b	1b	1-2b
kLUT	422.1	414.4	428.302
BRAM	846	1442	1320.5
Time (ms)	0.66	0.14	0.05
kFPS	1.53	7.28	18.31
Througput (GOPS)	1872.00	10108.00	21042.36
FPS/kLUT	3.62	17.57	42.75
GOPS/kLUT	4.43	24.39	49.13

Compared with Zhao's implementation, our implementation achieve 11.08x-11.97x improvements in performance (FPS), resource efficiency (FPS/kLUT) and throughput (GOPS). FBNA binarized all data include input and its' bit-width is only 1 bit, which brought a performance boost. However, unified compute unit cause decrease of efficiency when performing computation in different layer. Our implementation is more efficient computation which is 2.52x, 2.43x and 2.01x more efficient in performance, resource and power compared to FBNA.

And we further compare with FP-BNN which have almost the same power consumption, we still get a 2.38x speedup and 2.36x more power efficient as shown in Table IV. These results prove that our design is suitable for high throughput scenarios in terms of performance, resources and energy efficiency.

TABLE IV
COMPARISON OF POWER EFFICIENCY

	FP-BNN [15]	our-32PE
FPGA Board	Stratix-V 5SGSD8	Kintex XCKU115
Clock(MHz)	150	150
kFPS	7.69	18.31
Througput(GOPS)	9396.41	21042.36
Power(W)	26.20	26.40
FPS/W	293.60	693.58
GOPS/W	358.64	797.06

VIII. CONCLUSION

In this paper, we proposed a scalable fully pipelined architecture for binarized neural network. Our design focuses on higher throughput and resources utilization with pipeline technology. We further optimized our design by three optimization methods to achieve greater resource efficiency. Moreover, we proposed a design space exploration methodology based on analyses of models and resource utilization, to find the best degree of parallelism for each level with limited FPGA resources. Using the proposed methodology, we implemented six different configured BNN accelerators on Xilinx Kintex UltraSclae XCKU115 FPGA chip and achieved 2.24x-11.24x performance and 2.43x-11.79x resource efficiency improvement compared with prior BNN accelerators.

ACKNOWLEDGMENT

This work is supported by National Science and Technology Major Project 2018ZX01028101.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and GE. Hinton. Imagenet classification with deep convolutional neural networks. InAdvances in neural information processing systems, pages 1097-1105, 2012.

[2] O. Abdel-Hamid, AR. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu. Convolutional neural networks for speech recognition. IEEE/ACM Transactions on audio, speech, and language processing. Jul 2014.

[3] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and F.F. Li. Large-scale video classification with convolutional neural networks. InProceedings of the IEEE conference on Computer Vision and Pattern Recognition(CVPR), pages 1725-1732, 2014.

[4] Y. Cheng,D. Wang, P. Zhou, and T. Zhang. A survey of model compression and acceleration for deep neural networks. arXiv preprint arXiv:1710.09282, Oct 2017.

[5] R. Alvarez, R. Prabhavalka, and A.Bakhtin. On the efficient representation and execution of deep acoustic models. arXiv preprint arXiv:1607.04683, Jul 2016.

[6] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. The Journal of Machine Learning Research 18(1):6869-98, Jan 2017.

[7] M. Courbariaux, Y. Bengio, and J.P. David. Binaryconnect: Training deep neural networks with binary weights during propagations. InAdvances in neural information processing systems, pages 3123-3131, 2015.

[8] 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, Feb 2016.

[9] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. InEuropean conference on computer vision pages 525-542, Oct 2016.

[10] X. Lin, C. Zhao, and W. Pan. Towards accurate binary convolutional neural network. InAdvances in Neural Information Processing Systems, pages 345-353, 2017.

[11] W. Tang, G. Hua, and L. Wang. How to train a compact binary neural network with high accuracy?. InThirty-First Association for the Advance of Artificial Intelligence conference (AAAI), Feb 2017.

[12] R. Zhao, W. Song, W. Zhang, T. Xing, J.H. Lin, M. Srivastava, R. Gupta, Z. Zhang. Accelerating binarized convolutional neural networks with software-programmable fpgas. InProceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays(FPGA), pages 15-24, Feb 2017.

[13] P. Guo, H. Ma. R. Chen , P. Li, S. Xie, and B.Wang. FBNA: A Fully Binarized Neural Network Accelerator. International Conference on Field Programmable Logic and Applications (FPL), pages 51-513, Aug 2018.

[14] L. Yang, Z. He, and D. Fan. A fully onchip binarized convolutional neural network fpga impelmentation with accurate inference. InProceedings of the International Symposium on Low Power Electronics and Design, pages 1-6, Jul 2018.

[15] S. Liang, S.Yin, L. Liu, W. Luk, and S. Wei. FP-BNN: Binarized neural network on FPGA. Neurocomputing. 31;275:1072-86, Jan 2018.