2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)

A Cost-Effective CNN Accelerator Design with Configurable PU on FPGA

Chi Fung Brian Fong, Jiandong Mu, and Wei Zhang
Department of Electronic and Computer Engineering
{cfbfongaa,jmu}@connect.ust.hk, wei.zhang@ust.hk
Hong Kong University of Science and Technology

***Abstract*—Convolutional neural networks (CNNs) are rapidly expanding and being applied to a vast range of applications. Despite its popularity, deploying CNNs on a portable system is challenging due to enormous data volume, intensive computation, and frequent memory access. Hence, many approaches have been proposed to reduce the CNN model complexity, such as model pruning and quantization. However, it also brings new challenges. For example, existing designs usually adopted channel dimension tiling which requires regular channel number. After pruning, the channel number may become highly irregular which will incur heavy zero padding and large resource waste. As for quantization, simple aggressive bit reduction usually results in large accuracy drop.**

In order to address these challenges, in this work, firstly we propose to use row-based tiling in the kernel dimension to adapt to different kernel sizes and channel numbers and significantly reduce the zero padding. Moreover, we developed the configurable processing units (PUs) design which can be dynamically grouped or split to support the tiling flexibility and enable efficient hardware resource sharing. As for quantization, we considered the recently proposed Incremental Network Quantization (INQ) algorithm which uses low bit representation of weights in power of 2 format, and hence is able to represent the weights with minimum computing complexity since expensive multiplication can be transferred into cheap shift operation. We further propose an approximate shifter based processing element (PE) design as the fundamental building block of the PUs to facilitate the convolution computation. At last, a case study of RTL-level implementation of INQ quantized AlexNet is realized on a standalone FPGA, Stratix V. Compared with the state-of-art designs, our accelerator achieves 1.87x higher performance, which demonstrates the efficiency of the proposed design methods.

***Index Terms*—convolutional neural networks, FPGA, hardware acceleration**

I. Introduction

Deep neural networks (DNNs) have become a de facto standard for the next smart computing era. Convolutional neural network (CNN), as one of the most important types of the DNNs, has drawn the attention of many researchers since they play an important role in areas like real-time objection detection, audio speed recognition, etc [1], [2]. However, deploying CNNs on a portable system is challenging due to its computation and data-intensive nature. In order to address this challenge, methods for model reduction have been proposed, such as model pruning and quantization.

Channel pruning, which prunes the weights in filter level, is gaining attention as one of the most important pruning schemes. Plenty channel pruning strategies [3], [4] have been proposed to prune the channels with minimum accuracy degradation and computational complexity. However, the channel pruning may lead to an irregular channel distribution, thus resulting in difficulties in designing hardware for the pruned model. Currently, most hardware accelerators for CNN inference on FPGA use tiling in channel dimension but it requires regular channel numbers, such as multiple of 16 [5]–[7]. However, for the pruned network with irregular channel distribution, such tiling method may result in a large number of hardware idling (zero paddings) and resource waste.

A way to address the challenge is to reconsider tiling in kernel dimension along with output dimension since the number of different kernel size is usually very limited, not like the pruned channel number which can be arbitrary. Hence, tiling in kernel dimension can be carefully optimized to save more resources. There were previous works proposing 2D spatial tiling in the kernel [8], [9], however, to support different kernel sizes, computing a large size kernel from a fixed small kernel will introduce hardware idling at two edges and also create difficulties in hardware reuse for fully-connected (FC) layer. To resolve these hardware inefficiencies and compatibility issues, we proposed to use 1D tiling to replace the 2D tiling in the kernel dimension to best reduce the hardware idling and improve the resource efficiency and performance. Furthermore, an expandable computation architecture and configurable Processing Units (PUs) are developed to work together with the 1D tiling to enable the resource reuse for different layers with different kernel size.

Besides model pruning, quantization algorithm with low bit width has also been proposed as another way to reduce the hardware resource requirement, such as BNN [10], XNOR-Net [11] and DoReFa-Net [12]. They use very low bits, which is around 1-5 bits, to represent weights, the activations, and gradients so that the computing complexity and the buffer size can be greatly reduced. However, simple aggressive bit width reduction may incur large accuracy drop by $>10\%$. Different from above algorithms, one algorithm named as Incremental Network Quantization (INQ) proposed by Zhou *et al* uses less aggressive low bit width weight (5 bits) to represent the weight in power of 2 format, which can even achieve $>0.13\%$ accuracy improvement in commonly used network, such as AlexNet, VGG, Resnet. Besides, with this weight representation, the multiplication operation can be transformed

978-1-7281-3391-1/19/$31.00 ©2019 IEEE
DOI 10.1109/ISVLSI.2019.00015

into shift operation, and significantly facilitate the hardware implementation. Hence, to employ this benefit, we propose a novel approximate shifter design as the building block of PU to significantly save the resources and improve the frequency under only a slight loss of accuracy.

To summarize, this paper proposes a cost-efficient shifting-based CNN accelerator design with the following contributions:

- Propose row-based 1D-tiling to reduce hardware idling as well as gaining hardware flexibility.
- Propose an expandable computation block and configurable PU design to support different kernel sizes and enable hardware sharing between CONV and FC layers.
- Propose a novel approximate shifter to reduce the FPGA look-up table (LUT) usage without sacrificing much of accuracy.
- Implement a large-scale shifting-based CNN AlexNet accelerator [13]on Altera Stratix-V 28 nm FPGA with the proposed design methods as a case study.

Experiments demonstrate that we achieve an average throughput of 226.89 GOP/s for the CONV layers in AlexNet and 214.254 GOP/s for whole network, which is 1.87 times higher compared to the state-of-art works [5]–[8]. We can also achieve 24.6 GOP/s/W energy efficiency which makes our design suitable for embedded devices.

II. Background

A. Basic CNN

A typical CNN is composed of different layers, such as CONV, pooling, activation, FC and normalization. Different layers in a CNN try to extract features from the input. The extracted features are then categorized as a finite number of output classes. Among all layers in a typical CNN, CONV layers demand for the most computing power. The general spatial CONV algorithm is described in (1) and formed by six "For" loops.

$$out(f^+, y, x) = \sum_{f=0}^{N_f} \sum_{k_y=0}^{K} \sum_{k_x=0}^{K} W(f^+, f, k_y, k_x) \times F(f, y + k_y, x + k_x) \quad (1)$$

As for the activation layers, one of the typical activation algorithms is rectified liner unit (ReLU) which introduces nonlinearity and basically converts all the negative activation terms into zero. Additionally, the pooling layers also known as down-sampling, can be either followed by or come after the activation layer. One typical pooling algorithm is max pooling (MaxPool), which only remains the maximum value in each kernel. Between the CONV layers, normalization such as cross channel normalization (LRN) may be used. Recent networks prefer batch normalization (BN) as BN usually has a better performance. Finally, the fully connected layers receive the features from the previous layers and do the classification work.

To illustrate, one CNN example, AlexNet, is shown in Fig. 1. Note CONV layer 1 requires a stride of 4 while the other CONV layers require a stride of 1.

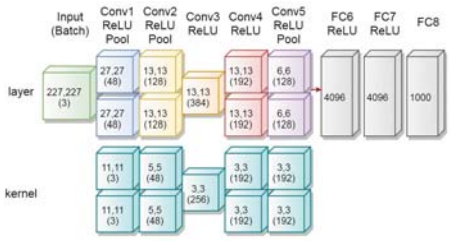


Fig. 1: CNN - AlexNet structure. Each layers boxes represent the output channel width, height and (length) either after the ReLU or MaxPool [13].

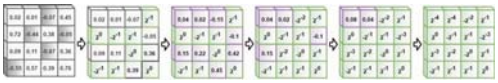


Fig. 2: INQ quantization process: quantize from left to right and the percentages are Raw $\rightarrow$ 50% $\rightarrow$ 50% after finetune $\rightarrow$ 75% $\rightarrow$ 87.5% $\rightarrow$ 100% [14].

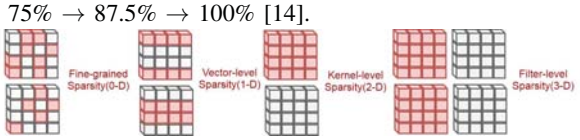


Fig. 3: Pruning at different dimensions, red color indicates pruned weights.

B. Channel Pruning

CNN is powerful at the cost of high computational complexity and intensive data accessing which inhibit it from a wider utilization. As one of the solution, channel pruning [15], which prunes the network in filter granularity, has been investigated in depth to make the neural network compact while maintaining its high performance so that the model can be fitted in power and resource constrained scenarios. Fig. 3 illustrates the pruning of the channels. Though simple, such pruning strategies have been widely deployed since it is easy to be implemented and no special hardware is needed to mask the pruned weights out.

Many channel pruning strategies have been proposed to pruning the channels while minimizing the accuracy loss. For example, [15] prune the channels by sparsity constraints. [4] conduct channel selection and parameter optimization in an iterative way. The resulted channel number of each layer will vary depending on the pruning strategies and may become irregular, which will raise difficulties for the hardware designs.

$$W_{inq} = \{\pm 2^{n_2}, ..., \pm 2^{n_1}, 0\} \text{ where } n_2 > n_1, \{n_1, n_2\} \in \mathbb{N} \quad (2)$$

C. Incremental Network Quantization Algorithm

To represent the weight efficiently with low bit width and resolve the accuracy loss at the same time, INQ algorithm [14] tries to quantize the well-trained weights to powers of 2, as illustrated in (2), iteratively with different percentages of quantization in each round. Fine-tuning will be conducted after each round to improve the accuracy. This process is illustrated

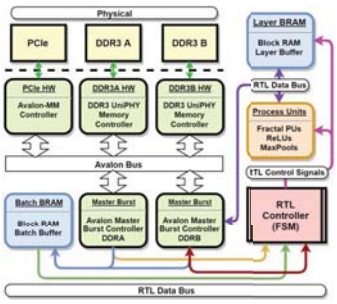
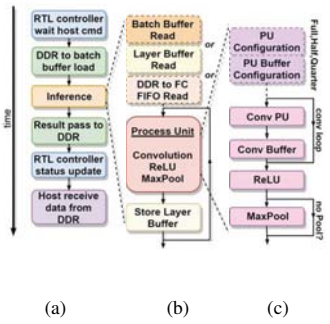
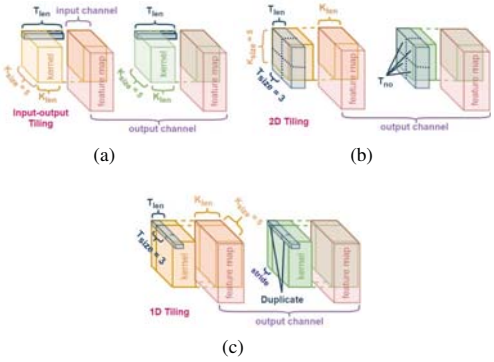


Fig. 4: Top block diagram



(a) (b) (c)

Fig. 5: (a) system timing diagram, (b) inference flow, and (c) Processing unit flow diagram



(a) (b)

(c)

Fig. 6: (a) Input-output channel tiling, (b) 2D tiling, (c) 1D row-based tiling.

in Fig. 2. The feed-forward loss is calculated during the fine-tuning process to indicate whether CNN training is converged during the weight quantization [14].

III. Hardware Design

A. System Overview

The system implementation is as illustrated in Fig. 4. The system flow, inference flow, and processing unit (PU) flow are shown in Fig. 5a, Fig. 5b, Fig. 5c respectively. Our computation block is based on PUs and each PU is built by a number of Processing Elements (PEs). The outputs of the PU are connected to the corresponding PU buffer which helps buffer the temporary results during the PU process. Then, the final result of the layer is stored in the layer buffer as the input of the next layer. We have implemented the convolution, ReLU, and MaxPool layers. To simplify the hardware design, the normalization layers are not implemented in this design as the impact of the LRN has limited improvements in the CNN accuracy [16]. Lastly, the FC layers share the same hardware resources with the CONV layer without adding excessive hardware and will be discussed next.

The input of the CNN is a colorful image and can be any size and aspect ratio. In order to fit the first CONV layer dimension, linear sizing ensures the size is set to 227x227 when AlexNet is applied. The weight of the INQ trained CNN is extracted and is stored in the BRAM with the sign value and the shift value, which can significantly reduce the BRAM size.

We allocate a dedicated type of memory to the inputs, outputs, and weights buffer. Three major types of memory are used, which are BRAMs, BROMs, and FIFOs. Since CONV weight size is relatively small, it is preferred to program them into on-chip BROM as much as the resource allows [17], while the FC weights are loaded into DRAM and pass to FIFOs when needed. As for the input data, they are firstly stored into DRAM and then passed to BRAM, and vice versa for output data.

TABLE I: Input-output channel tiling

	No Pruned							Pruned						
conv	Tlen	Klen	Ksize	output size	output length	set	total zero padding #	Tlen	Klen	Ksize	output size	output length	set	total zero padding #
1	32	3	11	55	48	2	1019M	32	3	11	55	40	2	849M
2	2x32	48	5	27	128	2	74.6M	2x32	40	5	27	40	2	35M
3	3x32	256	3	13	192	2	0	3x32	80	3	13	141	2	6.8M
4	5x32	192	3	13	192	2	0	5x32	141	3	13	60	2	4.3M
5	2x32	192	3	13	128	2	0	2x32	60	3	13	94	2	1.1M

TABLE II: 1D row-based tiling

	No Pruned							Pruned						
conv	Tsize	Ksize	Klen	output size	output length	set	total zero padding #	Tsize	Ksize	Klen	output size	output length	set	total zero padding #
1	4x3	11	3	55	48	2	9.58M	4x3	11	3	55	40	2	7.98M
2	2x3	5	48	27	128	2	44.8M	2x3	5	40	27	40	2	11.6M
3	3	3	256	13	192	2	0	3	3	80	13	141	2	0
4	3	3	192	13	192	2	0	3	3	141	13	60	2	0
5	3	3	192	13	128	2	0	3	3	60	13	94	2	0

B. 1D Tiling and Performance Analysis

Paralleling in input and output channel dimension is a commonly deployed hardware design for CNN inference accelerators since the number of the channels are usually regular and can be tiled neatly. However, this assumption doesn't hold for the first layer and the model after channel pruning, thus a zero padding becomes necessary for the input-output channel tiling as Fig. 6a illustrates. The notations are defined as tile number (T_{no}), size of each tile (T_{size}), length of the tile (T_{len}), kernel size (K_{size}), and kernel length (K_{len}). Table I shows two zero padding examples using a tile length of 32 on a pruned [18] and non-pruned AlexNet. The idling (zero padding) increases for the pruned network as the tile length 32 is not a common divisor of the input-output channel anymore, hence, always zero padded.

$$idle_{(PE\ 1D)} = ((T_{no} \times T_{size}) - K_{size}) \times T_{len} \times Duplicate \quad (3)$$

To better handle various channel numbers, especially the short channel number, tiling in kernel dimension can be an alternative approach. Previous works ever used 2D tiling in kernel dimension as Fig. 6b illustrates. However, implementing convolutions with a large kernel by numbers of small tile may result in 2 edges of idling (zero padding).

To reduce the zero paddings and better adapt to different kernel sizes, we propose to use row-based 1D tiling design,

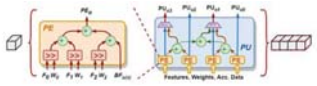
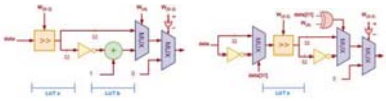


Fig. 7: PU to PE hardware design: (left) PE hardware, (right) PU hardware.

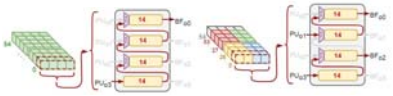


(a) (b)

Fig. 8: Shifter design: (a) standard, (b) approximate.

(a) (b) (c)

Fig. 9: PU configuration and mapping: gray blocks are idled, other colors represent parallelism in OC



(a) (b)

Fig. 10: PU to PU buffer connection examples: (a) one tap setting, (b) two taps setting.

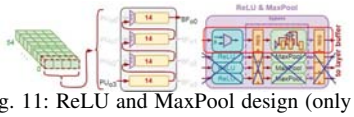


Fig. 11: ReLU and MaxPool design (only 1 set is needed).

Fig. 12: PU to layer buffer connection and design.

which can decrease the idling numbers by a quarter or more. The 1D tiling is illustrated in Fig. 6c. We process kernels in a row by row manner with a T_{len} = 1. Kernel rows for different output pixels might be loaded into the PEs at the same time so that all the PEs can be utilized efficiently. This further reduces the idling chance by shortening the tile length. Then the idling units for the 1D tiling can be derived in (3). Using the same non-pruned and pruned example, the new 1D row-based tiling zero padding numbers do not increase as Table II illustrated.

C. Design of Computation Blocks

To implement the row based 1D tiling design on hardware while fully utilize the computing power that the hardware has brought, we parallel our design in 3 dimensions, which is kernel row dimension (KR), featuremap row dimension (FR), and output channels dimension (OC).

Computation Block, which is the main computing array for our 1D tiling algorithm to achieve the aforementioned parallelism in 3 dimensions, is based on arrays of PUs. To support different kernel sizes and dynamically mapping the 1D row based tiles, a configurable PU design is developed which contains multiple PEs and the PU can be split according to the layer hyperparameters. The detailed PU, PE design, and PU configuration will be introduced in the following subsections.

1) PU and PE Design: The configurable PU is composed of four PEs as shown in Fig. 7 (right). The four PEs can work together to generate one sum of four PE outputs, or split into two groups and generate two sums, or output the 4 results separately. It is controlled by setting the multiplexers to select the corresponding adder outputs. This configurability enables different task groups to share one PU and greatly improves the hardware efficiency. Note that MUX is usually integrated into the FPGA LUT and DSP, hence will not cost extra resources.

As shown in Fig. 7 (left), one PE is composed of 3 shifters and an adder tree of 3 nodes. This is because the smallest kernel size of AlexNet is 3. The featuremaps are multiplied with weight by the shifters, then added with bias. Finally, the results are accumulated through the adder tree. We set the PE number to be 4 in one PU because the largest AlexNet kernel is 11, thus one PU can cover the kernel row with only minor PE idling. When considering different network, the PU, PE parameter can be adjusted accordingly.

2) Shifter Design: Since shifters and adders are fundamental units of the PEs, efficient designs for them play a critical rule in determining the hardware performance. Although adders can be implemented by the carry-chain inside of FPGA, barrel shifters are not integrated on-chip. As a result, LUTs are used to construct the barrel shifters efficiently. Notice that there are several challenges in the shifter design: Firstly, in order to maximize the throughput, it is demanded to put a large number of shifters within the constraints of the FPGA chip, however, this may cause routing congestion and a potential frequency decrease. Secondly, the latency of the shifter also plays an important rule in determining the operating frequency. The complex tradeoff between the parallelism and frequency makes the shifter design non-trivial. A general shifter design is shown in Fig. 8a. The multiplication of the feature and the weight is first conducted, then followed by a negative or positive value selection. A bypass for the case that all weight bits are zeros is also provided. The output will be directly set to zero in such cases.

To further save the LUT usage while minimizing the accuracy loss, we propose a novel approximate shifter as shown in Fig. 8b, where the adder is replaced by a logical shifter. The approximate shifter first checks the sign of the input feature. If the weight is positive, there will be no approximation. Otherwise, the approximate shifter only performs the inversion of the shifted data and omits the sum with one bit 1. The error introduced is only 1 LSB apart from the standard shifter. However, about 30% of the LUT and adders can be saved. Hence, the congestion problem of the final design routing can be greatly eased and the frequency can be significantly improved.

3) PU Configuration: As we previously mentioned, we are paralleling in KR, FR, and OC dimensions, which varies for different layers. As a result, the commonly used design strategy, which uses fixed number of PUs according to the fixed unrolling factors becomes infeasible. Instead, we proposed a novel design which can dynamically group the computation tasks into one PU, so that layers with different kernel sizes

and featuremap size can be processed efficiently.

To illustrate our strategy, we consider the representative layers in AlexNet. For the first conv layer, the input size is 227x227 and the output size is 55x55 with a kernel size of 11. we allocate all the 4 PEs within one PU to process the 11 shift-accumulation (SAC) for the KR parallelism since each PE can process 3 SAC in parallel. Totally 55 PUs are allocated for the FR dimension parallelism. OC equals to 1 in this case. This is illustrated in Fig. 9a. Similarly, for the second conv in AlexNet, we have a featuremap size of 27 and a kernel size of 5. As a result, 2 PE are allocated for KR parallelism. Considering each PU has 4 PEs, this offers a chance of splitting one PU to host 2 groups, which parallel in OC dimension. Finally, the 55 PUs can host the parallelism of 27 from the FR dimension and another parallelism of 2 in OC dimension. This is illustrated in Fig. 9b. Similarly, a conv layer with the output size of 13 and the kernel size of 3 is shown in Fig. 9c. In this case, the PU is further split to host four task groups.

To keep in pace with the PU design, the depth of the PU buffer must be larger or equal to 55 PUs. However, directly allocating 55 for the buffer size may not be hardware efficient. So we divide the PU buffer into taps. As for the PU buffer tap number and depth, similar design rules are used. Take AlexNet as an example, the PU grouping for the first 5 layers are 1x55, 2x27 and 4x13. In this case, the possible shallowest PU buffer is set to 14. To elaborate the PU buffer, Fig. 10 example shows how the PU buffer is connected, configured and designed.

4) ReLU, MaxPool and FC layer Design: Comparing with the convolutional layer, both activation and pooling hardware designs are straightforward. Each PE group in the PU can be processed with a dedicated ReLU and MaxPool hardware after the PU buffer. However, this is not resource efficient since N copies of activation and max pooling layer are needed, where N is the number of the taps. To further reduce the resource usage, we shifted out the data in the PU buffer sequentially, consequently, 1 out of N sets of ReLU and MaxPool is needed. The ReLU and MaxPool connections are shown in Fig. 11.

For FC layer, it also uses the same PUs for the convolutional layer to save the resource usage, however, in a different grouping scheme. In order for the FC layer to reuse the same PU, the kernel size is tiled to 6 when AlexNet is deployed, hence multiple output batches are processed concurrently. The kernel tiling size of 6 is because the first FC layer is exactly a multiple of 6.

5) Layer Buffer Design: To derive the depth of the layer buffer, the width of the output is helpful. For the first layer, the width is 27, because 55 feature outputs from CONV is processed by the MaxPool which has a stride of 2. The output width of the other layers are 13x2 (Layer 2), 13x4 (Layer 3, 4) and 4x6 (Layer 5). This indicates a layer buffer size of 27 is enough for most layers, unless 13x4, for which we use two separated sectors with a width of 27. To illustrate the connection between the PU and layer buffer, Fig. 12 shows how the layer buffer is mapped. As for the FC layer, FC uses the same layer buffer with CONV and the width is fixed to 13x4, which is the same with CONV 3 and 4.

TABLE III: ImageNet 2012 Accuracy Comparison

System	Software Caffe	Software Caffe	FPGA Proposed	FPGA Proposed
CNN Model	AlexNet[1] [13]	AlexNet [13] INQ [14]	AlexNet INQ[2]	AlexNet INQ
Top 1 accuracy no bias	-	53.492%	52.658%	52.684%
with bias	56.79%	55.703%	54.828%	54.834%
Top 5 accuracy no bias	-	77.225%	76.388%	76.416%
with bias	79.942%	78.829%	78.162%	78.170%

[1] with local response normalization (LRN) layer
[2] implemented with approximate shifter

IV. Experimental Result

We implemented a case study of 1D-tiling and shifting-based CNN accelerator of AlexNet on Altera Stratix-V DE5 board. Data is transferred from the host to the board via PCIe. The accuracy of both top 1 and top 5 are measured under 50,000 batch inference which is the same as the CNN accuracy measurement process in Caffe [19]. To compare with previous works, unfortunately, there is no hardware implementation of pruned AlexNet, so we implement the original AlexNet with proposed methods.

We considered the cases with and without weight bias, as it has an impact on BRAM usage, routing congestion and accuracy loss. The evaluation results of different approaches are shown in Table III. We compare the accuracy of our proposed design with the original AlexNet and INQ-AlexNet under the Caffe environment. The results show that our design manages to restrict the accuracy drop to $<1.3\%$. We can also observe from the table that the bias of the CNN improves the overall system accuracy by over 2%.

Table IV shows the results of comparisons with previous implementations on resource usage and throughput. The overall system throughput with the proposed design achieves 1.87x improvement compared to [5], [6], [8]. Compared with [7], for which the throughput is not given, our design has a similar latency. We also notice that, although the removal of bias together with approximate shifter may lead to a decrease in accuracy, they can greatly ease the routing congestion and improves the frequency.

We further illustrate the execution time by breaking it into 8 individual layers as shown in Fig. 13.

Power consumption is also crucial to the system performance. Unfortunately, DE5 does not provide an on-board current break-out for power measurement. To measure the power consumption, a hall effect clamp meter is applied to the PCIe 6 pin power cable to measure the current drawn from the FPGA, which also include all other unused components. Simultaneously, a high precision voltage meter is applied to the voltage rail of the PCIe power cable. With both voltage and current, power usage is calculated. However, such a power measurement is not as accurate as inline power measurement. For a fair comparison, the idle board power consumption without any design loaded on the FPGA board is measured and subtracted from the final power measurement. The unprogrammed DE5 power consumption is 12.857 W. The power consumption (after subtracting the unprogrammed

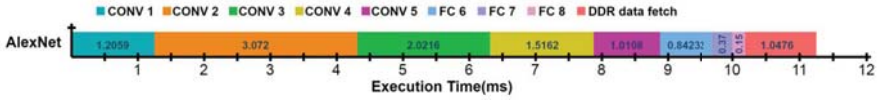


Fig. 13: AlexNet execution time breakdown. (CONV and FC time label includes ReLU and MaxPool execution time)

TABLE IV: Comparison of Previous Implementations

	AlexNet [5]	AlexNet [8]	AlexNet [6]	AlexNet [7]	This work: AlexNet[3,4]	This work: AlexNet[4]	This work: AlexNet[3]	This work: AlexNet
FPGA	Virtex-7 VX485T	Stratix-V GXA7	Stratix-V GXA7	Zynq XC7Z045	Stratix-V GXA7	Stratix-V GXA7	Stratix-V GXA7	Stratix-V GXA7
Design Entry	RTL + C	OpenCL	RTL	RTL	RTL	RTL	RTL	RTL
Base Clock Frequency	100 MHz	193.6 MHz	100 MHz	125 MHz	170 MHz	165 MHz	160 MHz	155 MHz
Number of operations per image	1.33 GOP	1.46 GOP	1.46 GOP	1.46 GOP	1.46 GOP	1.46 GOP	1.46 GOP	1.46 GOP
Number of weights	2.33 M	60.95 M	60.95 M	60.95 M	60.95 M	60.95 M	60.95 M	60.95 M
Precision	floating (32b)	fixed (16b)	fixed (8-16b)	fixed (16b)	fixed (32b)	fixed (32b)	fixed (32b)	fixed (32b)
DSP Utilization	2,240 (80%)	256 (100%)	256 (100%)	900	0 (0%)	0 (0%)	0 (0%)	0 (0%)
Logic Utilization[1]	186K (61%)	114K (49%)	121K (52%)	218.6k	137.5K (59%)	145.8K (62%)	146.9K (63%)	155.5K (66%)
On-chip RAM Utilization[2]	1,024 (50%)	1,893 (74%)	1,552 (61%)	Not Given	2,013 (79%)	2,013 (79%)	2,061 (81%)	2,061 (81%)
Convolution time	21.61 ms	19.86 ms	9.92 ms	8.22 ms	8.0476 ms	8.2915 ms	8.5510 ms	8.8265 ms
Fully-connected time	Not implemented	4.40 ms	2.83 ms	Not Given	1.2461 ms	1.2839 ms	1.3240 ms	1.3667 ms
Convolution throughput	61.6 GFOPS	67.5 GOPS	134.10 GOPS	161.98 GOPS	226.89 GOPS	220.22 GOPS	213.54 GOPS	206.87 GOPS
Overall throughput	NA	60.2 GOPS	114.50 GOPS	Not Given	214.254 GOPS	207.953 GOPS	201.651 GOPS	195.350 GOPS

[1] Xilinx FPGAs indicate as LUTs and Altera FPGAs indicate as ALMs.
[2] Xilinx FPGAs BRAMs based on (36 Kb) and Altera FPGAs based on M20K (20 Kb).
[3] CNN implemented with approximate shifter.
[4] CNN implemented without weight biasing.

power) for our design is 7.469 W when standby and 8.694 W when running the inference task.

Note that our design is based on LUTs. However, our design methods are general and can be applied to DSP based design as well.

V. Conclusion

In this paper, a shifting-based CNN inference design with configurable PU is demonstrated. Approximate computing is applied to a representative CNN algorithm, AlexNet, and manage to achieve system throughput and latency of 226.89 GOPs and 8.048 ms respectively. The experiments show 1.87x throughput and 20% latency improvement compared to an optimized RTL design on the same FPGA board [6]. Future work includes a low bit width FFT approach and a hardware-friendly normalization algorithm for DNN.

References

[1] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 1725–1732, 2014.

[2] O. Abdel-Hamid, A.-r. 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*, vol. 22, no. 10, pp. 1533–1545, 2014.

[3] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," *CoRR*, vol. abs/1707.06168, 2017.

[4] Z. Zhuang, M. Tan, B. Zhuang, J. Liu, Y. Guo, Q. Wu, J. Huang, and J. Zhu, "Discrimination-aware channel pruning for deep neural networks," *CoRR*, vol. abs/1810.11809, 2018.

[5] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 161–170, ACM, 2015.

[6] Y. Ma, N. Suda, Y. Cao, J.-s. Seo, and S. Vrudhula, "Scalable and modularized rtl compilation of convolutional neural networks onto fpga," in *Field Programmable Logic and Applications (FPL), 2016 26th International Conference on*, pp. 1–8, IEEE, 2016.

[7] S. I. Venieris and C.-S. Bouganis, "Latency-driven design for fpga-based convolutional neural networks," in *Field Programmable Logic and Applications (FPL), 2017 27th International Conference on*, pp. 1–8, IEEE, 2017.

[8] N. Suda, V. Chandra, G. Dasika, A. Mohanty, Y. Ma, S. Vrudhula, J.-s. Seo, and Y. Cao, "Throughput-optimized opencl-based fpga accelerator for large-scale convolutional neural networks," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 16–25, ACM, 2016.

[9] R. Morcel, H. Hajj, M. A. R. Saghir, H. Akkary, H. Artail, R. Khanna, and A. Keshavamurthy, "Feathernet: An accelerated convolutional neural network design for resource-constrained fpgas," in *ACM Trans. Reconfigurable Technol. Syst.*, vol. 12, pp. 6:1–6:27, ACM, 2019.

[10] 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.

[11] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European Conference on Computer Vision*, pp. 525–542, Springer, 2016.

[12] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

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

[15] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," *CoRR*, vol. abs/1608.03665, 2016.

[16] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[17] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "Shidiannao: Shifting vision processing closer to the sensor," in *ACM SIGARCH Computer Architecture News*, vol. 43, pp. 92–104, ACM, 2015.

[18] H. Mao, S. Han, J. Pool, W. Li, X. Liu, Y. Wang, and W. J. Dally, "Exploring the regularity of sparse structure in convolutional neural networks," *CoRR*, vol. abs/1705.08922, 2017.

[19] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*, pp. 675–678, ACM, 2014.