2019 32nd International Conference on VLSI Design and 2019 18th International Conference on Embedded Systems (VLSID)

UniWiG: Unified Winograd-GEMM Architecture for Accelerating CNN on FPGAs

Kala S*, Jimson Mathew†, Babita R Jose* and Nalesh S‡
*School of Engineering, Cochin University of Science And Technology, Kerala, India
Email: {kalas, babitajose}@cusat.ac.in
†Department of Computer Science, Indian Institute of Technology Patna, India
Email: jimson@iitp.ac.in
‡Department of Electronics, Cochin University of Science And Technology, Kerala, India
Email: nalesh@cusat.ac.in

***Abstract*—Convolutional Neural Networks (CNN) have emerged as the most efficient technique for solving a host of machine learning tasks, especially in image and video processing domains. However deploying CNNs on computing systems with smaller form factors have found to be extremely challenging due to the complex nature of CNNs. Hardware acceleration of CNNs using FPGAs have emerged as a promising approach due to high performance, energy efficiency and reconfigurability of FPGAs. Winograd filtering based convolution is the most efficient algorithm for calculating convolution for smaller filter sizes. In this paper, we propose a unified architecture named *UniWiG*, where both Winograd based convolution and general matrix multiplication (GEMM) can be accelerated using the same set of processing elements. This enables efficient utilization of FPGA hardware resources for accelerating all the layers in the CNNs. The proposed architecture has been used to accelerate AlexNet CNN, which shows performance improvement in the range of 1.4× to 4.02× with only 13% additional FPGA resources than state-of-art GEMM accelerator. We have also analyzed the performance with varying Winograd tile sizes and found out the most appropriate tile sizes for maximizing the performance while reducing on-chip memory resources.**

***Keywords*-CNN; Deep learning; FPGA; Machine learning;**

I. Introduction

Current decade has witnessed rapid explosion in the use of machine learning algorithms in a variety of application domains like robotics, computer vision, home automation and personal security. Development of more efficient algorithms, availability of large volumes of data for training and extensive amount of computing power provided by GPUs for accelerating these algorithms are the major drivers behind this advancement. Deep Neural Network (DNN) is one of the most widely used machine learning technique suited for tasks like image detection and recognition, speech processing and natural language processing. Convolution Neural Network (CNN) is more suited for image based applications. Most of the layers in CNN are not fully connected layers as in other deep learning architectures. Instead, images are divided into tiles and convolution operation is performed to each tile with a filter (kernel) tile. The filter coefficients or weights are common for all tiles in a single image vector. Hence the number of weights for a convolution layer is considerably less than an equivalent fully connected layer. Although CNNs provide considerable improvement in efficiency, considerable amount of processing is still required during training and the process can last for multiple days. GPU implementations are widely used to reduce the compute time. But high power consumption have rendered GPUs unsuited for embedded applications [1]. FPGAs are most commonly used hardware platforms which give decent power dissipation without compromising the performance.

Various FPGA implementations of CNN from different research groups are presented in [1], [6]–[10], [12], [13]. CNNs are composed of convolutional layers (CONV), max pooling (POOL), ReLU and fully connected (FC) layers. CONV layers constitute 90% of total computations in CNN [13]. Direct convolution, FFT-convolution and Winograd minimal filtering are the different algorithms for performing convolution [3], [4]. Direct convolution is performed using General Matrix Multiplication (GEMM) algorithm and is the least efficient of the three. However this approach has the advantage of using same hardware resources for CONV and FC layers. FFT based convolutions are suited for kernels of large size or if input data and kernel are of same size [4]. Most of the popular CNNs like VGGNet, AlexNet use small size kernels like 3×3 or 5×5 except for first CONV layer in Alexnet [2], [14]. Winograd algorithm is efficient for CONV layers when kernel size is small. Efficiency and precision of the algorithm depends on the Winograd tile size.

Motivation

Various implementations of Winograd Filtering based convolution have been proposed during the last three years [1], [6]–[8], [12], [13]. All of them use specially designed dedicated Processing Elements (PEs) for performing Winograd based convolution. A major drawback of this approach is the need for separate PEs for CONV layers and FC layers which lead to severe under-utilization of resources. As mentioned earlier, Winograd algorithm is suited only for CONV layers with small kernel sizes. In case of AlexNet, only CONV2, CONV3, CONV4 and CONV5 layers can be efficiently accelerated using Winograd algorithm. CONV1 will need direct convolution. If we can perform direct convolution, Winograd convolution and FC layer computation using the same compute resources, we can achieve significant savings in hardware.

2380-6923/19/$31.00 ©2019 IEEE
DOI 10.1109/VLSID.2019.00055

In this paper, we propose an architecture for accelerating CNNs on FPGAs which incorporates Winograd filtering algorithm on a GEMM accelerator. Such a unified architecture gives the most efficient implementation for CONV layers irrespective of the kernel sizes and also for FC layers. For this purpose, we make use of the technique proposed in [3], where the Winograd based convolution is transformed to GEMM operations. The technique can be incorporated to any efficient GEMM accelerator. For demonstrating the technique we have used the FPGA based GEMM architecture in [11].

Major contributions of this paper are:

- We have proposed a unified Winograd-GEMM architecture for CNNs where both Winograd filtering and GEMM can be performed using the same array of processing elements.
- We implement the proposed architecture in FPGA targeting AlexNet CNNs.
- We study the trade-off between on-chip memory and performance for AlexNet for this architecture.

Rest of this paper is organized as follows. Section II gives background and related works. Winograd minimal filtering is explained in Section III. Section IV describes the proposed unified architecture. FPGA implementation and results are explained in Section V. Section VI concludes the paper.

II. Background and Related Work

In a typical CNN, feature extraction is performed by CONV layers, sub sampling is done by POOL layers and FC layers perform classification. Direct convolution is a multiply-and-accumulate operation where each input element is multiplied with each of the kernel elements and results are summed up. Consider C channels of $K \times K$ kernels, and a total of F such kernels. Also consider C channels of $H \times W$ input feature maps. Each kernel performs convolution on the input feature map with a stride of S, and generates the output feature map, Y. Element (u, v) in f^{th} kernel, c^{th} channel is represented as $G_{f,c,u,v}$ and input element $(\overline{x}, \overline{y})$ in c^{th} channel of i^{th} image is denoted as $D_{i,c,\overline{x},\overline{y}}$. Each output element is given by

$$Y_{i,f,\overline{x},\overline{y}} = \sum_{c=1}^{C} \sum_{u=1}^{K} \sum_{v=1}^{K} D_{i,c,\overline{x}+u+S,\overline{y}+v+S} \times G_{f,c,u,v} \quad (1)$$

Complete convolution output can be written as in Equation (2),

$$Y_{i,f} = \sum_{c=1}^{C} D_{i,c} * G_{f,c} \quad (2)$$

where * denotes correlation.

In [1], an end-to-end FPGA CNN accelerator is proposed with all layers working in a pipelined fashion. Efficient bandwidth utilization for FC layers with minimum off-chip memory is the major focus in [1]. An FPGA based accelerator exploiting all levels of parallelism in CNNs is presented in [12]. [13] implements a compiler that can analyze the CNN structure and parameters, and automatically generate modular and scalable compute resources for accelerating these CNNs. Another compilation tool for mapping CNN to FPGA has been proposed in [10] along with a data quantization strategy which helps to reduce the bit-width down to 8-bit with negligible accuracy loss. Above mentioned architectures use matrix-matrix multiplications for computing convolutions. In [4], the complexity of direct and FFT convolutions are evaluated and GPU implementation of FFT based convolution network is implemented. However FFT based convolution techniques are efficient only for larger kernels. For smaller kernels, Winograd algorithm can be used which can significantly reduce computational complexity in terms of multiplications. Winograd based CNN accelerators have been proposed in [3], [5], [6]. For efficient CNN implementations, a hybrid approach is required whereby appropriate convolutions algorithms are chosen for each layer based on the image and kernel size.

III. Winograd Minimal Filtering

A 2D Winograd filter is represented by $F(m \times m, r \times r)$ where $m \times m$ is the output size and $r \times r$ is the kernel size. The kernel is applied on an input tile size of $(m + r - 1) \times (m + r - 1)$. For a kernel tile g and input tile d, output Y can be written as,

$$Y = A^T[(GgG^T) \odot (B^T dB)]A \quad (3)$$

where G, B and A are transform matrices for kernel, input and output respectively. These matrices can be precomputed, once the value of m and r are known. For $F(2\times2, 3\times3)$ tile, direct convolution involves $2^2 \times 3^2 = 36$ multiplications, whereas Winograd takes only $4^2 = 16$ multiplications. Multiplication complexity is reduced by a factor $\frac{36}{16} = 2.25$. In general, the ratio of hardware complexity in terms of multiplication for conventional and Winograd convolution is given by $\frac{m^2 \times r^2}{(m+r-1)^2}$ As Winograd tile size increases (ie., m increases for a particular r), number of additions and constant multiplications also increase. The size of transform matrix also increases with tile size and the coefficients in transform matrices need approximation which reduces the precision of convolution operation. However multiplication complexity reduces with increase in tile size. Hence there is a clear trade-off between precision and performance based on Winograd tile size.

A. Winograd Filtering Using GEMM

Equation (3) calculates the convolution of image and one kernel for a single channel. For multiple channels, convolution outputs from all channels have to be added together to get the final output as in Equation (2). This can be expressed as

$$Y_{i,f,\overline{x},\overline{y}} = \sum_{c=1}^{C} D_{i,c,\overline{x},\overline{y}} * G_{f,c} \quad (4)$$

$$= A^T \left[\sum_{c=1}^{C} (Gg_{f,c}G^T) \odot (B^T d_{i,c,\overline{x},\overline{y}} B) \right] A \quad (5)$$

Substituting $U = GgG^T$ and $V = B^T dB$ which are the transformed kernel and image tile respectively,

$$Y_{i,f,\overline{x},\overline{y}} = A^T \left[\sum_{c=1}^{C} U_{f,c} \odot V_{c,i,\overline{x},\overline{y}} \right] A \quad (6)$$

By reducing the image tile coordinates $(i, \overline{x}, \overline{y})$ to a single dimension and separating each element-wise multiplication in a tile, the inner *add* and *multiply* in Equation (6) can be transformed to GEMM [3]. Here GEMM is between the transformed input tiles and the kernel tiles. Each element of input tile and kernel tile after transformation is scattered to a different matrix. Transformed image matrix consists of corresponding elements from consecutive image tiles for a single channel along the column and one column for each channel. Transformed kernel matrix which is arranged as channels along the rows and kernels along the columns.

The complete algorithm has the following steps. First, transform the image and kernel tiles and scatter the elements to different matrices. Next, matrix multiplication is applied to corresponding matrix pairs and the elements are gathered back and output transform is performed. Detailed algorithm for GEMM based Winograd filtering is presented in Section IV. By transforming Winograd filtering to GEMM, a unified architecture can be used for accelerating all layers of CNN. The unified architecture will consist of programming elements for performing GEMM and additional modules for performing input, kernel and output transforms. We call this unified architecture as **UniWiG**.

IV. Proposed Unified Winograd-GEMM Architecture

Implementing GEMM on FPGA requires a trade-off between performance, external memory bandwidth and BRAM utilization. Typically, blocked variants of GEMM algorithm are implemented which optimally use the BRAM resources and DDR bandwidth while maximizing performance. Similar blocking techniques have to be applied to GEMM based Winograd algorithm.

A. Blocked Winograd Filtering Algorithm

Basic GEMM based Winograd filtering algorithm has been presented in [3]. For efficient FPGA implementation, we have modified this algorithm to create a blocked Winograd Filtering algorithm which is listed in Algorithm 1. Basic algorithm in [3] transforms the Winograd operation into q×q matrix multiplications where $q = m + r - 1$ is the input tile size. Each multiplication is between matrices of size T×C and C×F where T, C and F are number of input tiles, channels and kernels respectively. In Algorithm 1, first matrix is divided into sub-matrices of size $S_T \times C$ and second matrix into sub-matrices of size $C \times S_F$. Input and kernel transforms are performed at sub-matrix level, GEMM is performed on the transformed sub-matrices, and resultant product sub-matrix is transformed to get S_T output tiles (each of size $m \times m$) for S_F kernels. This process is repeated for all tiles and filters. Hereafter we denote F$(m \times m, r \times r)$ as F(m, r).

Algorithm 1: Blocked Winograd Filtering Algorithm for $F(m \times m, r \times r)$

$T = \lceil H/m \rceil \lceil W/m \rceil$ is the number of image tiles.
$q = m + r - 1$ is the input tile size.
Each input tile is $q \times q$ and tile stride is m .
$d_{c,t} \in R^{q \times q}$ is input tile t in channel c.
$g_{f,c} \in R^{r \times r}$ is filter f in channel c.
G, B^T and A^T are filter, data and output transforms.
S_T and S_F are the row and column block size.
for $\tau = 0$ to $\lceil T/S_T \rceil$ **do**
  for $\phi = 0$ to $\lceil F/S_F \rceil$ **do**
    for $f = \phi * S_F$ to $(\phi + 1) * S_F$ **do**
      for $c = 0$ to C **do**
        $u = G g_{f,c} G^T \in R^{q \times q}$
        Scatter u to matrices $U : U^{\xi,\nu}_{f,c} = u_{\xi,\nu}$
      end for
    end for
    for $t = \tau * S_T$ to $(\tau + 1) * S_T$ **do**
      for $c = 0$ to C **do**
        $v = B^T d_{c,t} B \in R^{q \times q}$
        Scatter v to matrices $V : V^{\xi,\nu}_{c,t} = v_{\xi,\nu}$
      end for
    end for
    for $\xi = 0$ to q **do**
      for $\nu = 0$ to q **do**
        $Z^{\xi,\nu} = U^{\xi,\nu} V^{\xi,\nu}$
      end for
    end for
    for $f = 0$ to S_F **do**
      for $t = 0$ to S_T **do**
        Gather z from matrices $Z : z_{\xi,\nu} = Z^{\xi,\nu}_{f,t}$
        $Y_{f,t} = A^T z A$
      end for
    end for
  end for
end for

B. Architecture

In Algorithm 1, calculating the matrix M using GEMM is the most compute intensive operation. This can be performed using any efficient GEMM accelerator. Input tiles d have to be transformed using $B^T dB$ and filter tiles g have to be transformed using GgG^T before passing to the GEMM accelerator. Similarly products from GEMM z has to be transformed using $A^T zA$. Filter coefficients are constant for a given CONV model and hence the filter transform can be applied off-line.

For our unified architecture, we have selected the state-of-art multi-array accelerator in [11] for performing GEMM. We have modified the accelerator by including additional modules for performing input and output transforms. Filter transforms are performed as off-line computations and the transformed filter coefficients are stored in DDR. Fig. 1 gives the high level block diagram for proposed unified Winograd-GEMM (**UniWiG**) architecture. The two additional blocks namely *Data Transform Unit (DTU)* and *Output Transform Unit (OTU)*, are introduced between the *Memory Access Control* and the PE arrays. Outputs from the *DTU* are sent

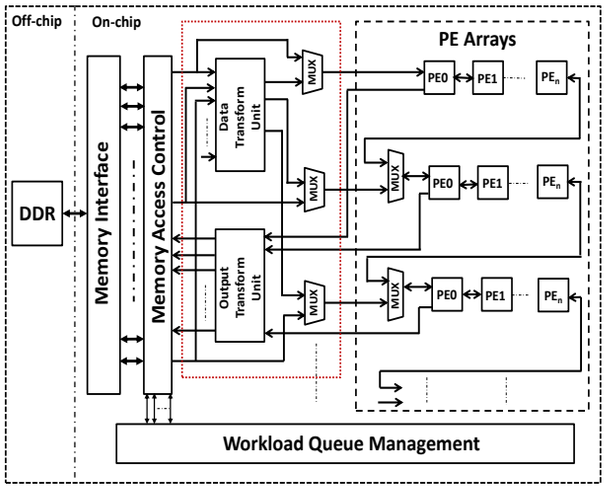

Fig. 1: Proposed **UniWiG** Architecture

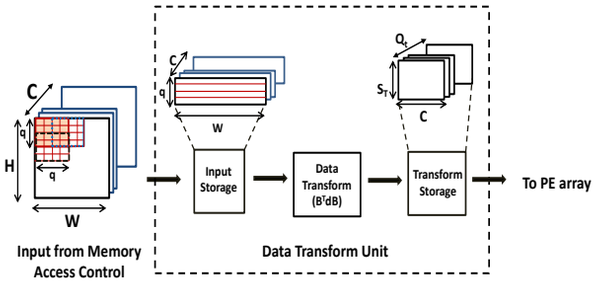

Fig. 2: Data Transform Unit

to the PE array. Transformed filter coefficient sub-blocks are fetched directly to the PE array. PEs perform the sub-block multiplication and send the output to *OTU*. This unit computes the operation $(A^T zA)$ which gives the final output. Additional multiplexers are introduced so that *DTU* and *OTU* can be bypassed to perform normal GEMM computations.

Fig. 2 shows the block diagram for *DTU*. Transform is applied to input tiles of size $q \times q$ and elements from q rows are required to create one tile. In order to optimally utilize the burst accesses from DDR, a complete row is fetched in one shot. q such rows from each channel are fetched and stored in the *Input Storage*. Data corresponding to one tile is read out of *Input Storage*, transformed and saved into *Transform Storage*. This is repeated for all tiles from all the channels. Stride length for tile formation is m which implies that only m additional rows need to be fetched for forming new tiles. Tiles are created as sub-blocks of size S_T. *Transform Storage* stores $Q_t = q \times q$ sub-matrices of size $S_T \times C$. The transform operation $B^T dB$ involves constant multiplications and can be expressed as a set of floating point additions to reduce the hardware complexity. *DTU* is fully pipelined so that new input rows can be fetched while current rows are transformed and send to PE.

PE array consists of multiple systolic arrays of processing elements. Neighboring arrays can be chained together to increase the array length. Longer array means lesser bandwidth

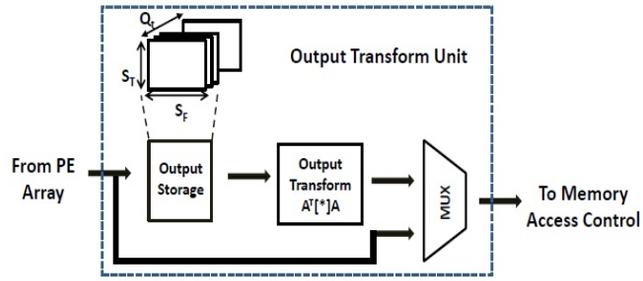

Fig. 3: Output Transform Unit

pressure on DDR. However matrix sub-block sizes have to be large enough to fill all elements of the array. Input sub-blocks to PE array are read from *Transform Storage* and filter sub-blocks are fetched directly from DDR. Each input sub-block is convolved with all kernel tiles in consecutive operations. Reuse of input sub-block ensures that input data transform need to be performed only once for each input tile.

Fig. 3 gives the block diagram for *OTU*. The Q_t sub-block products are stored in *Output Storage*. Corresponding elements from all Q_t sub-blocks are collected to reform the tile and output transform is applied by multiplying with A^T and A. This transform also involves constant multiplication and can be converted to additions. Each tile gives outputs of size $m \times m$ after the transform. These outputs are sent to *Memory Access Control* to be written back to the DDR. OTU and DTU are designed such that multiple Winograd tile sizes are supported. Depending on the chosen size, transform units can be dynamically reconfigured.

C. Block RAM Memory Requirement

We estimate the extra BRAM required for *DTU* and *OTU* based on input and filter parameters. *Input Storage* needs to store $2m$ rows of image data for each class C. *Transform Storage* stores Q_t matrices of size $S_T \times C$ while *Output Storage* stores Q_t matrices of size $S_T \times S_F$. We use single precision floating point data so that total additional BRAM locations required is $4 \times (2mWC + Q_t S_T C + Q_t S_T S_F)$ bytes.

D. Performance Model

In [11], authors have developed an analytical performance model for GEMM on multi-array architecture. This model can be modified to estimate the performance of Winograd algorithm on our proposed modified multi-array architecture. Let N_p be the number of PE arrays working in parallel. Average number of sub-block multiplications for Winograd performed in one array is given by

$$N_{work} = \lceil \frac{Q_t}{N_p} \times \lceil \frac{T}{S_T} \rceil \times \lceil \frac{F}{S_F} \rceil \rceil \tag{7}$$

Given BW, the external memory bandwidth in bytes, average time taken in seconds to load a pair of sub-blocks is given by

$$T_{work} = \frac{4(mWC \frac{S_T}{\lceil \frac{W}{m} \rceil} \times \frac{1}{\lceil \frac{F}{S_F} \rceil} \times \frac{1}{Q_t} + S_F C + S_T S_F)}{BW} \tag{8}$$

TABLE I: AlexNet CONV Layer Parameters for $F(2,3)$

Parameter	CONV1	CONV2	CONV3	CONV4	CONV5
Input(H×W)	227×227	27×27	13×13	13×13	13×13
Kernel(r×r)	11×11	5×5	3×3	3×3	3×3
Kernels(F)	96	256	384	384	256
Channels(C)	3	48	256	192	192
Tiles (T)	-	196	49	49	49

First term in the numerator in Equation (8) gives the cycles to transfer extra rows of image data for generating S_T transformed tiles. Q_t such matrices are computed from a single input tile and the transformed image sub-blocks are reused for $\frac{F}{S_F}$ filter sub-blocks. Second term gives the cycles for transferring the filter sub-blocks and the third term gives the cycles for output write-back. Total time for data transfer is

$$T_{trans} = N_{work} \times T_{work} \tag{9}$$

Computation time in seconds for a PE array is given by

$$T_{comp} = \frac{N_{work}(S_F + Max(S_T, S_F)C + Stage_{fmac})}{F_{acc}} \tag{10}$$

Here $Stage_{fmac}$ gives the pipeline stage in PE and F_{acc} is the clock frequency of the accelerator.
Bounds for execution time is given as

$$Max(T_{comp}, T_{trans}) < T_{total} < (T_{trans} + T_{comp}) \tag{11}$$

If data transfer and compute are completely overlapping, then lower bound of execution time can be achieved.

V. Implementation Results and Analysis

We have implemented proposed unified architecture on Xilinx XC7VX690T FPGA. The same FPGA platform as that used in [11] has been selected for easier performance comparison. Convolution layers in AlexNet CNN is accelerated using the implemented architecture. Table I shows the parameters for CONV layers in AlexNet. CONV1 layer uses a kernel of size 11×11 and is computed using direct convolution. Rest of the layers are computed using the Winograd algorithm. For CONV2, we have used the technique proposed in [10] to convert the 5×5 filters to four 3×3 filters. Hence all four layers have filter size of 3×3.

Implementation in [11] has four arrays of 64 PEs each. Multiple arrays can be dynamically tied together in *cooperative mode* to form larger arrays. N_P is the number of independent arrays. Detailed analysis has been done to decide block size and N_P for each layer. Reducing N_P (having larger arrays) reduce the required DDR bandwidth since all PEs in one independent array shares the same memory interface. However, for maximum PE utilization, block size has to be equal to number of PEs in each independent array.

In the blocked Winograd algorithm, the two matrices to be multiplied are of size $T \times C$ and $C \times F$. From Table I, for the last three CONV, maximum tile size is only 49 which means that in a 64 PE array, some of the PEs will always remain idle. Hence we have changed the PE array configuration to eight arrays of 32 PEs each and use only the *independent mode* in case of blocked Winograd algorithm. Note that, the total number of PEs remain same as in [11] and all the configuration used in [11] is possible by interconnecting multiple arrays.

TABLE II: Estimated BRAM for Various Tiles

CONV Layer	F(2,3)	F(3,3)	F(4,3)	F(6,3)
CONV2	41	62	83	141
CONV3	126	197	376	603
CONV4	98	153	292	468
CONV5	98	153	292	468

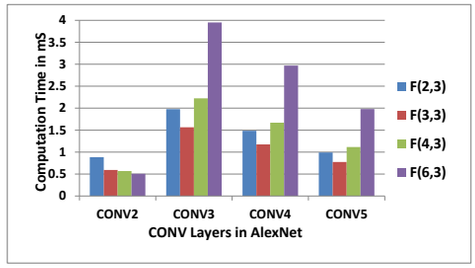

Fig. 4: Compute time for CONV layers using various tiles

As already mentioned, more number of independent PE arrays leads to higher DDR bandwidth requirement. However input to PE arrays are provided by the *DTU* and efficient input feature reuse strategies have been employed to mitigate this issue. Also each Winograd operation is converted to Q_t GEMM operations, all of which can be performed in parallel. Hence for all the CONV layer computations using Winograd algorithm, we have fixed N_P as 8. Sub-block size is another parameter that affects the performance. Sub-block sizes S_T and S_F have to be closely matched for maximum performance. Unbalanced blocking can result in more time spend on loading the larger block to the PEs. This is clear from Equation (10) where computation time depends on $Max(S_T, S_F)$. However since we are operating all arrays in *independent mode*, we will fix S_F to 32, number of PEs in each array and S_T to a value close to 32 which reduces the number of sub blocks, T/S_T.

In [11], N_P and sub block size are the two parameters that are leveraged to extract the optimum performance for each layer. We have already fixed these two parameters for Winograd based convolutions. However in case of Winograd algorithm, size of Winograd tile is an additional parameter which affect the performance. Increasing the tile size reduces computations and thus increases the performance. However larger tile sizes demand more BRAM for the three storages mentioned in Section IV-B. Also larger tile size implies lower precision. If lower precision levels are acceptable, we can use the tile size to trade-off between BRAM requirement and performance. We have used four tile sizes namely $F(2,3)$, $F(3,3)$, $F(4,3)$ and $F(6,3)$ [3].

For the selected tile sizes, we have used performance equations from Section IV-D to calculate the expected performance. BRAM sizes can be predicted using equation from Section IV-C. Predicted additional BRAM requirement are in Table II. Fig. 4 shows expected compute time for one convolution with

TABLE III: Performance comparison

	GFLOPS		CONV/sec	
CONV Layer	Proposed Work	[11]	Proposed Work	[11]
CONV2	61.53	87.8	1581.21	392.22
CONV3	72.6	64.9	592.2	217.07
CONV4	72.25	64.1	786.05	571.77
CONV5	72.84	62.9	1188.60	841.60

TABLE IV: Resource Utilization

Resource	Proposed Work	[11]	Percentage Increase
BRAM	757.5	560.50	13.4
DSP48E	1123	1032	2.46
Flip Flops	383412	292016	10.51
LUTs	252336	192493	13.8

varying tile sizes for CONV2 to CONV5 layers. We have used lower bounds for expected compute time.

Fig. 4 shows that compute time is minimum using $F(3,3)$ and increases for larger tile sizes. These layers have smaller image feature sizes so that large tile sizes result in increased zero padding. With this tile size, maximum BRAM requirement is 197 for CONV3. For CONV2, maximum performance is for tile $F(6,3)$ and BRAM requirement is within 197. Thus we select $F(6,3)$ for CONV2 and $F(3,3)$ for all other layers with number of BRAMs fixed as 197.

With this configuration, we have performed Winograd based convolution for the 4 AlexNet layers on our architecture. Table III shows performance comparison for these layers with [11]. In [11], performance is reported in terms of GFLOPs. However since Winograd algorithm leads to reduction in computation, performance comparison with direct convolution techniques using GFLOPs is not fair. In such cases, number of convolutions per second is a better metric for comparison. Table IV gives FPGA resource utilization for the proposed architecture in comparison with [11]. Table III shows that our architecture gives considerable improvement in performance in terms of convolutions/second for all four layers. CONV2 layer shows maximum improvement of 4.02× against expected performance of 5.06× for $F(6,3)$ Winograd tiles. For other layers performance ranges from 1.4× to 2.72×. Table IV shows that very less additional FPGA resources are used for introducing Winograd algorithm. Only 13.8% of the total LUT resources and 13.4% of total BRAMs are required. We are able to achieve significant performance improvement with very few additional hardware resources which shows the effectiveness of our approach. Note that proposed architecture can compute direct convolutions and GEMM operations with the same performance as reported in [11]. In terms of GFLOPs, performance is higher for all layers except CONV2. Reduced performance for CONV2 indicate that the blocking strategy employed for this layer is not efficient enough to fully exploit the reduction in computations from using $F(6,3)$ tile. Table V compares the performance of proposed architecture with other state-of-art FPGA based CNN accelerators. Among floating point implementations, only [12] shows higher performance. The difference in performance is only 5%. As mentioned earlier GFLOPs is not the correct metric to compare architectures using different convolution algorithms. However these results show that proposed unified architecture gives the same level of performance as other state-of-art floating point based implementations.

TABLE V: Comparison of Various CNN Implementations

	Network	Precision	FPGA	F(MHz)	Performance
Ours	AlexNet	32-float	XC7VX690T	200	79.54 GFLOPS
[8]	AlexNet	32-float	VX485T	100	61.62 GFLOPS
[7]	AlexNet	16-fixed	XC7Z045	150	187.8 GOPS
[5]	AlexNet	32-float	XC7VX690T	200	50 GFLOPS
[9]	custom	48-fixed	SX240T	200	16 GOPS
[12]	AlexNet	32-float	VX485T	100	84.2 GFLOPS
[15]	custom	fixed	VLX240T	150	17 GOPs

VI. Conclusions

This paper presents a novel unified architecture for implementing both GEMM algorithm and Winograd minimal filtering algorithm on the same datapath. These algorithms form majority of computations involved in all major CNNs. Proposed architecture has been implemented in FPGA and uses multiple arrays of processing elements. With just 13% additional hardware resources, this implementation gives 1.4× to 4.02× performance when compared to state-of-art architecture. AlexNet CNN model has been implemented using the proposed architecture and shows better average performance when compared to other recent architectures. Proposed architecture is a significant addition to the variety of CNN implementations available in literature.

References

[1] Huimin Li et.al., "A High Performance FPGA-based Accelerator for Large-Scale Convolutional Neural Networks". In *FPL* 2016.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". *NIPS* 2012.

[3] Andrew Lavin and Scott Gray. "Fast Algorithms for Convolutional Neural Networks". In *IEEE CVPR*,2016.

[4] Michael Mathieu, Mikael Henaff, and Yann LeCun. "Fast Training of Convolutional Networks through FFTs". In *ICLR 2014*.

[5] Roberto DiCecco, et.al., "Caffeinated FPGAs: FPGA Framework For Convolutional Neural Networks". In *IEEE FPT* 2016.

[6] Liqiang Lu et.al, "Evaluating Fast Algorithms for Convolutional Neural Networks on FPGAs". In *IEEE FCCM*, 2017.

[7] Jiantao Qiu et.al, "Going Deeper with Embedded FPGA Platform for Convolutional Neural Network". In *ACM/SIGDA FPGA* 2016.

[8] Chen Zhang, et al., "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks". In *ACM/SIGDA FPGA* 2015.

[9] Srimat Chakradhar, et al., "A Dynamically Configurable Coprocessor for Convolutional Neural Networks". In *ACM ISCA* 2010.

[10] Kaiyuan Guo, et.al, "Angel-Eye: A Complete Design Flow for Mapping CNN Onto Embedded FPGA". *IEEE TCAD* 37, 1 (Jan. 2018), 35–47.

[11] Junzhong Shen, et al., "Towards a Multi-array Architecture for Accelerating Large-scale Matrix Multiplication on FPGAs". *IEEE ISCAS* 2018.

[12] Mohammad Motamedi et.al., "Design Space Exploration of FPGA-Based Deep Convolutional Neural Networks". In *IEEE ASP-DAC* 2016.

[13] Yufei Ma, et al., "Scalable and Modularized RTL Compilation of Convolutional Neural Networks onto FPGA". In *FPL* 2016.

[14] K. Simonyan and A. Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In *ICLR 2015*.

[15] Maurice Peemen, et al., "Memory-Centric Accelerator Design for Convolutional Neural Networks". In *IEEE ICCD* 2013.