# FC_ACCEL: Enabling Efficient, Low-Latency and Flexible Inference in DNN Fully Connected Layers, using Optimized Checkerboard Block matrix decomposition, fast scheduling, and a resource efficient 1D PE array with a custom HBM2 memory subsystem

**Nick Iliev** ( niliev4@uic.edu )
University of Illinois at Chicago
**Amit R Trivedi**
University of Illinois at Chicago

**Research Article**

**Keywords:** Neural Network Accelerators, Fully Connected layers, Sparse-Dense Tensors, kernels, DNN, CNN, AlexNet, VGG-16, HBM memory, digital VLSI, Checkerboard matrix-vector operations scheduling

**Posted Date:** February 3rd, 2022

**DOI:** https://doi.org/10.21203/rs.3.rs-1321782/v1

# FC_ACCEL: Enabling Efficient, Low-Latency and Flexible Inference in DNN Fully Connected Layers, using Optimized Checkerboard Block matrix decomposition, fast scheduling, and a resource efficient 1D PE array with a custom HBM2 memory subsystem

NICK ILIEV, AMIT RANJAN TRIVEDI, University of Illinois, ECE Department, Chicago, USA

This article presents a novel low latency CMOS hardware accelerator for fully connected (FC) layers in deep neural networks (DNNs). The accelerator, FC-Accel, is based on 128 8x8 or 16x16 processing elements (PEs) for matrix-vector multiplication, and 128 multiply-accumulate (MAC) units integrated with 16 High Bandwidth Memory (HBM) stack units for storing the pre-trained weights. A dedicated non-blocking crossbar switch is not used in our low-latency page-bus demultiplexer-based interconnect between the 16 HBMs and the 128 PE array. We show near-linear speedup, reduction in space complexity, and reduction in time complexity with respect to traditional parallel matrix-vector multiplication with a checkerboard block decomposition algorithm, using a novel matched HBM2 memory subsystem for weights and input feature storage; we perform 16-bit fixed-point computation on the key kernels for DNN FC layer computation : FC kernel with KxM tiles which can be scaled for different FC layer sizes. We have designed a flexible processing element, PE, which implements the scalable kernel, in an 1D array of PEs to conserve resources. PE reconfiguration can be done as required by the layer being processed (FC6,FC7,FC8 in AlexNet of VGG16 for example). Micro-architectural details for CMOS ASIC implementations are presented and simulated performance is compared to recent hardware accelerators for DNNs for AlexNet and VGG_16. When comparing simulated processing latency for the FC8 layer, FC-Accel is able to achieve 108 GOPS (non-pipelined, with a 100 MHz clock) and 1048 GOPS (pipelined, with a 662 MHz clock) which improves on a recent EIE accelerator quoted at 102 GOPS with a 800 MHz clock and using compression for the same FC8 layer. When compared to Tensaurus, a recent accelerator of Sparse-Dense Tensor computations, FC-Accel (clocked at 662 MHz) delivers a 2.5 increase in throughput over Tensaurus (clocked at 2 GHz) for VGG16 FC8. The Xilinx Versal-ACAP VC1902 FPGA has an FC8 inferencing latency of 158 usec at 1.33 GHz, which is much slower than FC-Accel's FC8 latency of 8.5 usec. When compared with an NVIDIA Jetson AGX Xavier GPU running inference on VGG-16 FC8, FC-Accel reduces FC8 inferencing latency from the GPU's average of 120 usec to 8.5 usec. Intel's Arria-10 DLA FPGA achieves 26 usec for the VGG16 FC8 layer which is 3 times the latency of the proposed solution.

CCS Concepts: • **Computer systems organization → Embedded systems**; *HBM*; • **CNN Accelerators**;

Additional Key Words and Phrases: Neural Network Accelerators, Fully Connected layers,Sparse-Dense Tensors, kernels, DNN, CNN, AlexNet, VGG-16, HBM memory, digital VLSI, Checkerboard matrix-vector operations scheduling

## 1 INTRODUCTION

This research has been motivated by an important problem in real-time integrated hardware-software implementations of devices at the edge of the cloud : low-latency evaluation of fully connected (FC) layers for neural-network processing performed within the device. Example applications include deep CNN processing such as AlexNet or VGG-16, where a typical fully-connected layer has 4096 input features and 1000 output neuron activations. Another application is bounding-box object localization in an image using reinforcement learning for training and a Q-Network for inferencing with several fully-connected layers. Typical neural network hardware accelerators, such as Intel's Movidius, Habana-Goya, and Google's TPU dedicate specific micro-instructions and micro-architectural processing resources for FC layer evaluation. FC layer evaluation is usually

Author's address: Nick Iliev, Amit Ranjan Trivedi, niliev4@uic.edu, University of Illinois, ECE Department, Chicago, 851 S Morgan St, Chicago, Illinois, USA, 60607.

a dense matrix-vector multiplication problem of considerable size. As an example, AlexNet has an FC8 layer with 4096 input neurons and 1000 outputs (activations), which is similar to the FC8 layer in VGG-16. It has been shown [21] that dense FC layer evaluation is a major contributor to latency during DNN inferencing, when compared to the initial sparse convolutional layers. The large number of weights for FC layer evaluation must of stored in off-chip memory during training, and read from the same memory almost every cycle during inferencing, creating a severe memory bandwidth problem. Therefore recent research has focused on hardware acceleration of FC layers in particular with various attempts at reducing the memory bandwidth demands. This includes pruning or compression of some or all DNN layers which is not used in our approach since we want to achieve maximal accuracy during inferencing. Fig. 1 shows such an FC layer which is the focus of our work.

The evaluation of the FC layer in the figure, for one vector of input features, is formulated as a matrix-vector multiplication (tensor) problem as shown in Fig. 2.

Section 2 presents relation of our approach to prior works on hardware acceleration of FC layers in DNNs such as AlexNet and VGG-16. Sections 3 discusses the proposed micro-architecture for our FC-ACCEL design. Sec. IV presents simulation results. Section 4 discusses future extensions of the proposed micro-architecture. Sections 5 concludes.

## 2 RELATION TO PRIOR-ART

Hardware acceleration of DNNs has typically focused on both convolutional, CONV, and fully connected, FC, layers. This imposes some restrictions on the micro-architecture which has to handle both sparse, CONV specific kernels, as well as dense, weights based FC layers. Yuran et al. [24] accelerate FC and CONV layers with a common processing element, PE, which is based on a matrix multiplier. Convolutions are unrolled to matrix multiplications for the PEs to process. The same PEs have to acccelerate the FC layers as well which can create a resource contention problem. Our solution differs from this approach since we have PEs dedicated to the FC layers only, and the sizes of the FC weights tiles (sub-matrices) are not dictated by CONV kernel and loop-unrolling considerations. Instead our PEs are optimized to reduce latency processing of the FC layer and minimize number of passes to process the entire FC layer. Jiantao et al. [12] propose to compress the FC layer weights by using Singular Value Decomposition, SVD. This approach may not always work since SVD may not exist or be numerically stable for some large FC weights matrices. In his implementation PEs are shared for CONV and FC processing and are not optimized for FC layers specifically as in our proposed FC-ACCEL architecture. Ning et al. [16] present a global summation architecture to completely replace the matarix multiplications in the FC layers. A mathematical identity replaces multiplications with accumulators for each feature map. This places a large hardware resource requirement for FC layers with large feature maps; only small image sizes of 32x32 have been processed with the global summation method. In contrast, our FC-Accel can handle FC6 25088-4096 feature layers in VGG16. Huimin et al. [9] propose an accelerator PE for both CONV and FC layers, with a batch-based computing method for the FC layers only. This differs from FC-Accel which operated on the entire FC layer ( all feature maps) and uses tiles (batches) only for the weights matrix. Their solution also has to apply two different computing patterns on FC layers which is not needed in our approach : FC-Accel uses the same computing pattern for all FC layers. Li [15] proposes a PE architecture for matrix-vector multiplication in FC layers. An entire row of weights is fetched from off-chip memory for the PEs to process. FC-Accel fetches only tiles (sub-matrices) of weights from a given column for all PEs to process and processes all rows simultaneously, column by column. The recent NVIDIA Volta GV100 architecture [17] uses Tensor Cores for matrix arithmetic. HBMs [11] are used for weights and data storage. Each Tensor Core can complete 64 floating point mixed-precision operations per clock. FC-Accel computes 128

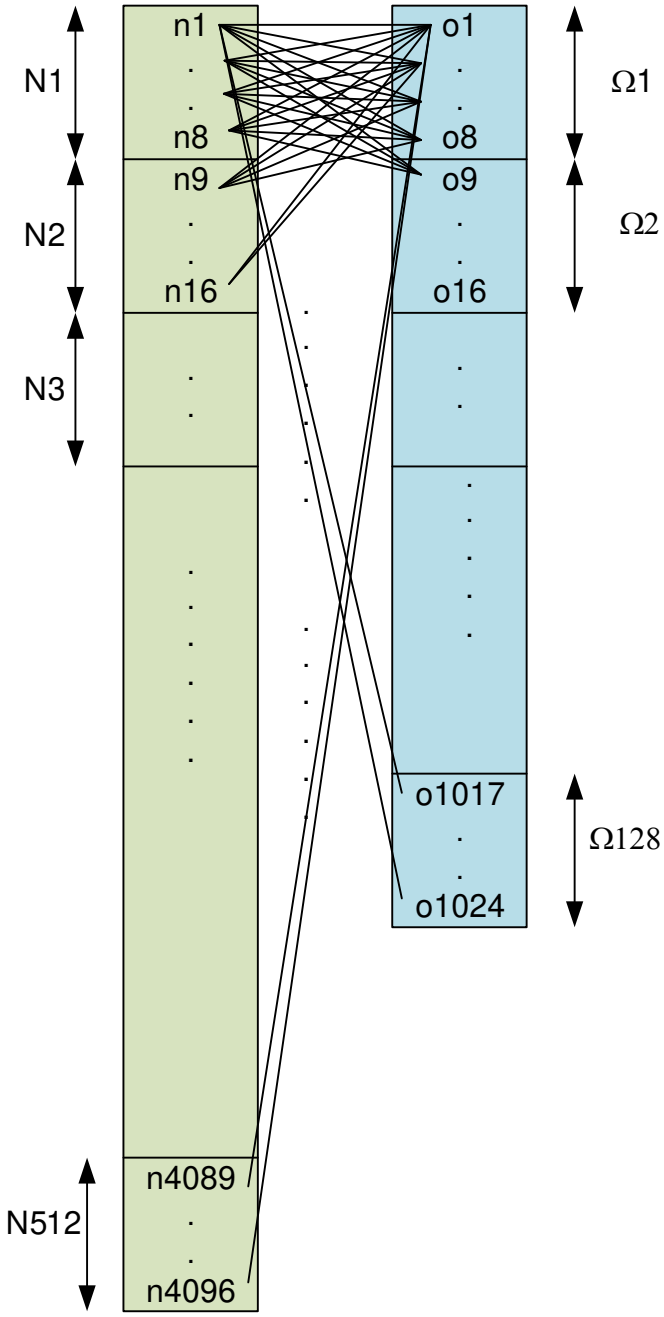


Fig. 1. Fully Connected FC8 layer in AlexNet or VGG-16 : 4096 input and 1000 outputs. Groups of 8 are indicated in the input and output vectors respectively.

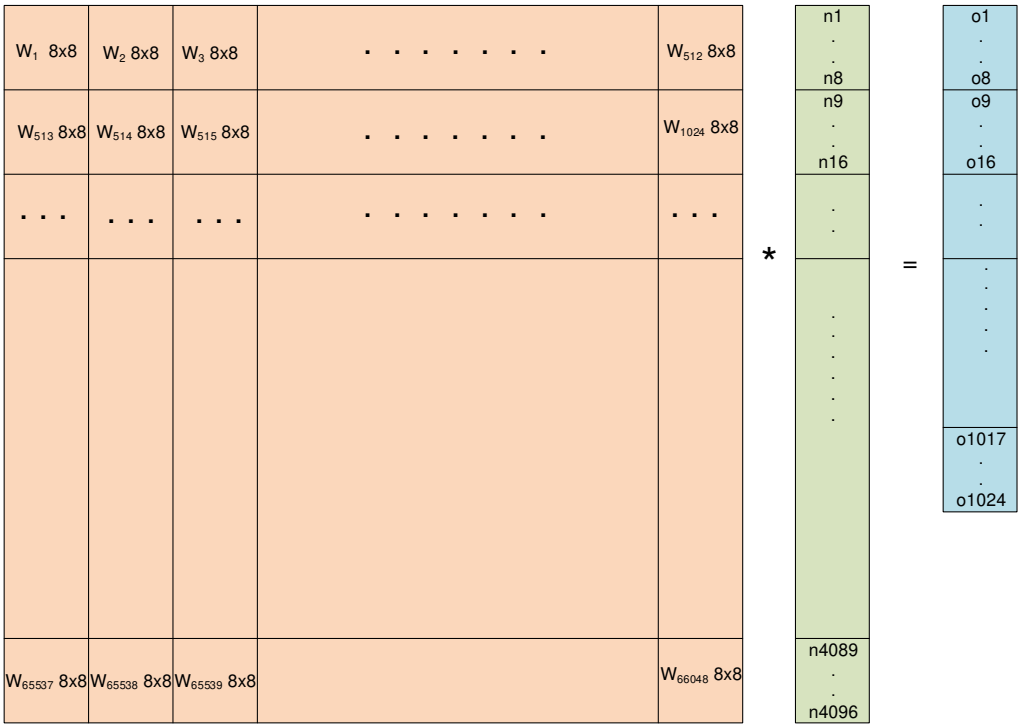


Fig. 2. The equivalent matrix-vector multiplication for the FC layer in Fig. 1. The weights are grouped in 8x8 sub-matrices (tiles) W1, W2, etc. Each column of sub-matrices is mapped, during its time-slot, to a set of 128 MACs and 128 PEs.The same set of MACs and PEs is re-used for all 512 time-slots during processing.

16-bit fixed point operations per clock. The NVIDIA Jetson AGX Xavier GPU with DLA (Deep Learning Accelerator) [18] has 64 Tensor Cores running at 1.37 GHz. When running inference on VGG-16 FC8, the GPU-DLA combination can achieve an average FC8 latency from 120 to 180 usec. In comparison, FC-Accel takes advantage of the KxK tile decomposition of the weights matrix and uses 128 PEs in parallel thus achieving a greater level of parallelism than the DLA.

The CNAPS ASIC [7] has a SIMD architecture with an array of 16x8 scalar multipliers for matrix-vector multiplication, MVM, while FC-Accel uses 8x8 or 16x16 arrays of scalar multiplier for MVM. The DianNao series of ASICs [2] implement an array of 64 16-bit integer MACs. FC-Accel uses 128 16-bit fixed-point MACs instead. The DaDianNao and ShiDianNao ASICs [4] store all weights on chip (eDRAM or SRAM) while FC-Accel uses on-chip HBMs with silicon interposers for storing weights for all FC layers and input features to these layers.

A dedicated non-blocking crossbar switch for interfacing HBM memory to a neural network accelerator has been proposed in [14]. This is an FPGA-based accelerator (Xilinx Alveo Versal with AXI interface between 3D-DRAM HBM and convolution and average-pooling engines) for randomly wired neural networks (RWNNs ) similar to ResNet-50, and does not target FC layers in AlexNet or VGG16. The largest non-convolutional (random) layer in the implemented RWNN has 32 nodes with 380,192 (436x872) parameters with is considerably smaller than the FC8 layer

in VGG16 or AlexNet. In comparison, our proposed interface to 3D-DRAM HBM is not restricted to AXI (Versal) transactions, does not require a crossbar switch, and allows for lower latency and more flexible pipeline for data pre-fetching.

The recent Xilinx Versal ACAP VC1902 FPGA, [23], has 96 AI Engines (AIEs) in its Deep Learning Processing Unit (DPU). When running at 1.33 GHz the DPU delivers an average FC8 inference latency of 158 usec. The larger level of parallelism in FC-ACCEL's 128 PE array, along with its optimal scheduling of each HBM read access, improves on this and delivers a lower FC8 latency, as well as lower FC6 and FC7 latencies.

Google's recently announced Edge Tensor Processing Unit, Edge TPU, [6] uses up to 65536 8-bit MAC units which limits forward inference to 8-bit precision. HBM is also used for weights and features storage. By contrast, FC-Accel maintains 16-bit fixed point precision in forward inference passes. The recently described EIE ASIC [8] accelerates both CONV and FC layers by using compression to derive a compressed network model. The resulting matrix-vector multiplications are of smaller dimensions however an 800 MHz processing clock is needed to achieve 102 GOPS for FC8 layer processing. In comparison, a non-pipelined FC-Accel at 100 MHz achieves 108 GOPS for the FC8 layer. A pipelined FC-Accel needs a 662 MHz clock for FC8 processing and achieves 1048 GOPS. The TETRIS DNN accelerator in [5] uses Hybrid Memory Cube, HMC, 3D DRAM memory which is an early form of HBM. Using 16 3D engines and 16 HMCs it can achieve 627 GOPS at 500MHz, which is 40 % lower than FC-Accel's performance.

The recent Tensaurus DNN ASIC accelerator, [20], uses a 2D array of PEs and an HBM memory to store all input freatures and weights. Each PE can compute a dot product of two vectors (VMUL), and addition of two vectors (VADD), in order to implement matrix-matrix and matrix-vector products. A crossbar switch is used for each column in the 2D PE array, and out-of-order HBM read accesses to each PE are needed to avoid blocking sequential memory accesses. In contrast, FC-ACCEL stores weights in 16 separate HBMs (providing 128 virtual page-bus channels of weights). Each page-bus channel of weights drives its dedicated PE in a 1D PE array, HBM read accesses are in order and non-blocking, and each PE computes a matrix(tile)-vector(tile) product instead of a vector dot product. When running VGG16 FC8, Tensaurus can achieve a 7x increase in throughput over an equivalent baseline Intel Xeon CPU implementation, or 420 GOPS at 2 GHz. FC-ACCEL achieves 1048 GOPS at 662 MHz for the FC8 layer.

Intel's Deep Learning Accelerator, DLA, [1], is used in Stratix-10 and Arria-10 FPGAs and is closest in micro-architectural details to the proposed FC-ACCEL with some major differences outlined in the following. Both DLA and FC-ACCEL use a 1D array of PEs for all processing. In DLA it's a systolic 1D array, while the proposed FC-ACCEL iterates (re-uses) each PE in its array and accumulates all intermediate results for the final activation stage. Each DLA PE performs a vector dot product operation to implement general matrix-matrix and matrix-vector multiplications; FC-ACCEL's PE performs a tiled matrix-vector operation, with a configurable tile size. FC layer processing is done in DLA by re-using the PE array which has been optimized for convolutions and not for dense matrix-vector multiplication. FC-ACCEL's PE and PE array is optimized for these dense operations. It can also be configured to perform 2D convolution with additional iterations over the 1D PE array as discussed in the Section "Future Extensions". The DLA completes VGG16 FC8 layer inferencing in 26 usec which is 3 times larger then FC-ACCEL.

## 3 FULLY CONNECTED LAYER ACCELERATOR ARCHITECTURE

We store all FC layer weights in 16 stacks of Hight Bandwidth Memory (HBM, see JESD235A/B standard [11] ) in order to maximize the memory bandwidth of each read-out access from the weights memories. Each HBM and its interposer read out 8 pages of weights, on 8 128-bit page read-out buses. Each page bus connects to its data-prefetch unit (DPR-BUF) which assembles a 1024

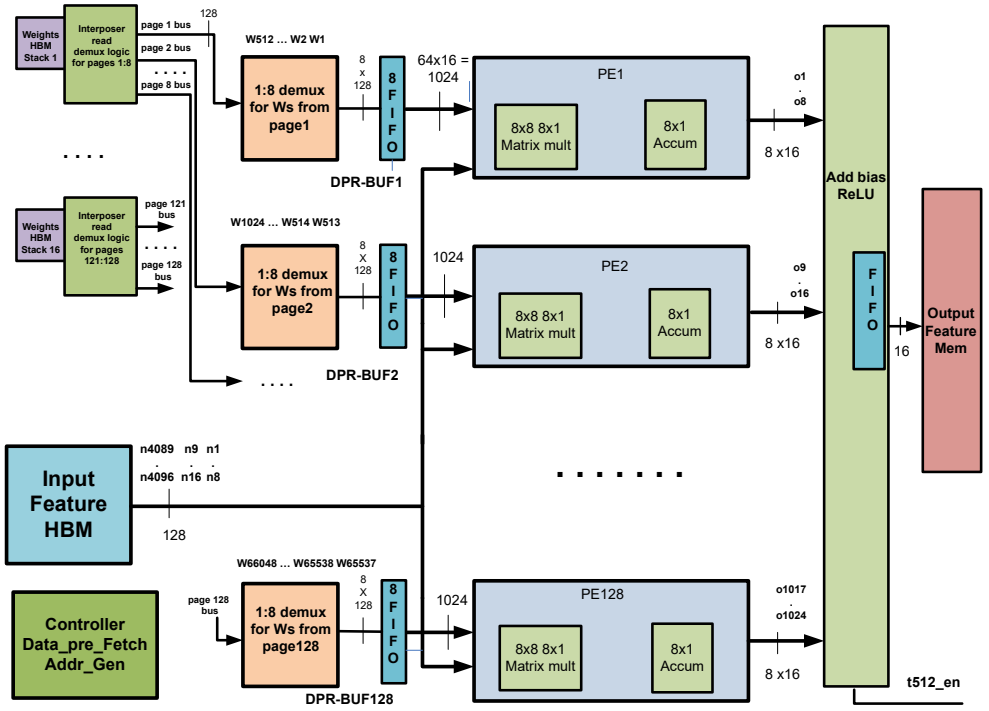


Fig. 3. High level architecture block diagram. Each HBM stack and interposer drives 8 128-bit page busses and each page bus has its dedicated data-pre-fetch unit and address generator. A DPR-BUF data-pre-fetch unit ensures that 1024 bits of weights are aligned for a single cycle read by its PE. Input and output memories have dedicated address generators. One top-level controller schedules the data flow in all 128 PE channels.

bits (64 x 16-bits) buffer of weights. This 1024 bits buffer allows the read out of 64 16-bit weights for each PE's matrix multiplier in 1 clock cycle.

Fig. 3 shows a high-level view of the proposed architecture. It implements a column of tiles decomposition of the original weights matrix. Each HBM and interposer connect to 8 DPR-BUF buffers for driving 8 PEs in parallel. Each DPR-BUF unit schedules a stream of two reads to two sequential column addresses so that a stream of 8 128-bit read bus cycles is generated. The following section on the data-prefetch unit details how 8 read operations from an HBM page fill 8 FIFOs contained in each DPR-BUF. The 16 HBM stacks connect to the 128 PEs via silicon interposers which are not shown. Each interposer has read addressing logic to read out 8 pages and to connect the HBM's 128-bit read bus to the corresponding page-bus. A dedicated non-blocking crossbar switch, as proposed in [14] is not used in order to minimize access latency. The 128 PEs are reused in each of the 512 time slots which map to the 512 columns of Fig.2. The weights matrix in Fig.2 is broken up in 8x8 tiles of weights, which dictates the 8x8 PE design. Accordingly the input data is divided up into tiles of 8 elements each. Other network sizes, multiples of 8x8, are therefore possible for example 512 columns and 512 rows (square matrix in Fig.2) or 4096 inputs and 4096 outputs (FC7 in AlexNet and VGG16), 25088 inputs and 4096 outputs (FC6 in VGG16) and so on. The following sub-sections detail the micro-architecture of each PE sub-block.

Our choice of an HBM dedicated to 8 PEs avoids the need for complex 2D mesh routing and the newtork-on-chip, NoC, hardware required to implement the routing infrastructure, as required

in [13] and in [5]. Note that we divide each 1024 bit wide interface into 64 16-bit independent channels ( not 8 128-bit independent channels as done in some HBM interposers) to match our PE bandwidth requirement ( 8x8 PE with 64 16-bit weights) exactly.

Notice that off-line training may produce several sets of weights (for several training optimality criteria, eg. acceptable loss function values) which can be stored in different pages in each HBM. During real time operation, between inferencing passes, a new page may be selected in some or in all HBMs and the FC layer will use a new set of weights for the next inference pass. Therefore HBM-based weights storage allows dynamic ( real-time ) weights selection between inference passes. In a following section "Up-Scaling to Larger FC Layers" we show how the proposed micro-architecture can be up-scaled for the larger FC6 and FC7 layers by using 128 16x16 PEs and their corresponding 16 HBMs.

### 3.1 HBM Data-Prefetch Unit, DPR-BUF

The weights tiles stored in an HBM contain a set of 64 16-bit two's complement values for a specific 8x8 matrix-vector multiplier (MV-mult). The scheduler has to drive all inputs of the MV-mult in one clock cycle during its scheduled time slot. The MV-mult inputs include 8 16-bit two's complement values of input features from HBM_IN (1 cycle 128 bits read out from HBM_IN) as well as 64 16-bit weight values, which form a 1024 bit parallel bus of weights to the MV-mult. The DPR-BUF ensures that this 1024 bit bus is driven by 8 128-bit bus outputs of each HBM as shown by 4 Da and 4 Db transactions in Fig.4. Note that an HBM's 8 DRAMs make up a stack and each DQ[127:0] output of a DRAM contributes to a portion of the DPR-BUF's 1024-bit on-chip buffer after being rate-matched by its FIFO. Two clock domains, a 500MHz wr_clk (write into FIFO), and a 662 MHz rd_clk (read from FIFO), are used in the DPR-BUF. This matches the HBM's 500MHz DQ[127:0] bus to the 662 MHz clock domain used in the pipeliend PEs and up to the ReLU's output FIFO write port.

Fig. 4 is from the JESD235C HBM2 standard and shows how 1024 bits can be read out with two read requests, using burst length of 4, BL4, with R=6 to two column addresses in the same bank. The two read requests generate 8 128-bit transactions on the DQ[127:0] bus which is sampled by the DPR-BUF. Following the main controller's sequence, the DPR-BUF initiates two read accesses to all HBMs during cycles T0 to T9 overlapped with a read access to the Input features memory HBM_IN for the next input value in order for them to align at the MV-mult interface. This is shown in the following Fig. 5.

The 8 128-bit read out cycles, in the 500 MHz clock domain, from an weights HBM (in BL4 mode) fill up its DPR-BUF's 1024-bit buffer. In the 662 MHz clock domain, the 1024 bit buffer is read in 1 cycle, Rd, overlapped with read out of the input from HBM-IN. The following 3 662 MHz cycles are processing cycles P1, P2, P3. If not empty, the FIFO is then read in the next Rd cycle and so on. In the 500 Mhz domain, the HBM is read in cycles m1 to m8. After each mx cycle, the FIFO is written in its corresponding wrx cycles. This is shown in Fig. 6. Note that cycle m8 is followed by cycle sw, to allow for HBM bank switching if the two read commands map to different banks. The main control sequence can allow for more sw cycles if needed.

### 3.2 Matrix-Vector Multiplier Unit

Each PE contains a dedicated 8x8 matrix-vector multiplier MV-mult for fixed-point data in the Q(17,10) format. The choice of an 8x8 tile in the weights matrix in Fig. 3 determines the size of the matrix-vector multiplier as well as the number of HBMs and PEs in the system. We use 8x8 tiles of weights as an example implementation and other sizes are possible in the proposed architecture as well. MV-mult contains an array of 64 scalar multipliers where both operands have the same bit width in the Q(17,10) format. Each product is also truncated and rounded to fit into Q(17,10). The selection of 17 bits from the total of 34 bits (before truncation) is configurable and can be decided

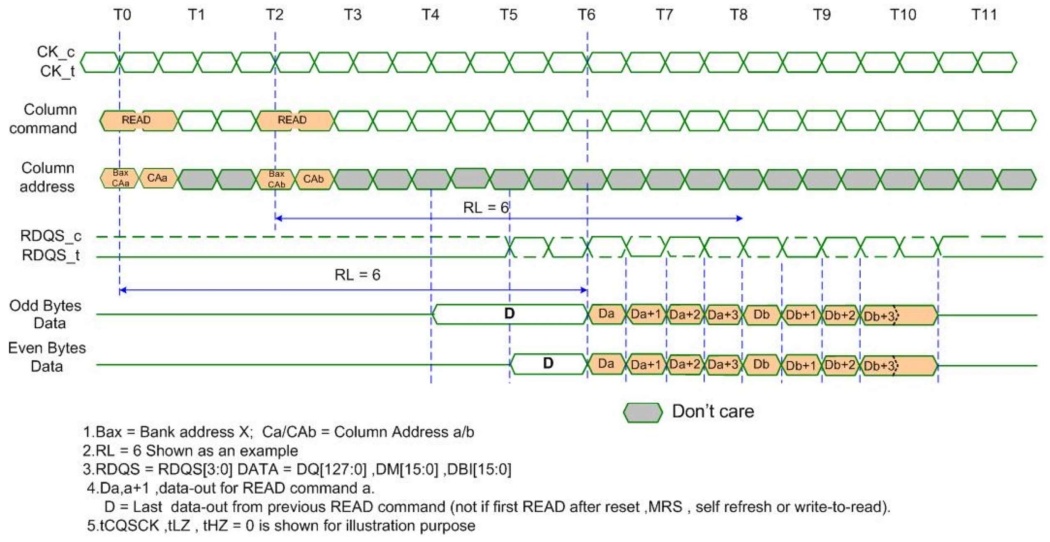


Fig. 4. Read access timing from JESD235C. The access starts with the first read request to column address Ca. After R T-cycles (T0 to T6 for R=6 example) a burst of 4 128-bit words, Da to Da+3, is available on DQ[127:0]. Similarly, the second read request to column address Cb generates a burst of 4 128-bit words, Db to Db+3. The DPR-BUF combines the 8 128-bit words and writes them into 8 corresponding FIFOs. The 8 FIFOs are then read into the 1024 bit on-chip buffer.

by the dynamic range of the FC layer from offline calibration. A two-stage pipeline is implemented by a dedicated register at the output of each scalar multiplier. An adder tree of seven Q(17,10) adders sums all partial products for each of the 8 rows. A zero-detector is used for each operand to gate off switching within the module when one or both operands are zero. The output 8x1 vector of products is available in 1 100 MHz clock cycle in an ASIC PDK 45 nm implementation. Fig. 7 shows the details of MV-mult. Note that for the pipelined PE ASIC implementation described later, the critical path in the seven adder tree is reduced to 1.51 nsec using a seven stage pipeline. This allowed us to run the pipelined design of a PE at 662 MHz and increase the max throughput of the accelerator considerably.

### 3.3 Vector Accumulator Unit

Each PE in Fig. 3 has an 8x1 vector accumulation unit, V-Accum, for adding up the partial products generated during each of the 512 time-slots. A V-Accum maps to each 8x1 row of the weights matrix in Fig.2; for example V-Accum-1 to o1-o8, V-Accum-2 to o9-o16 and so on. Each V-Accum receives the prod-1 to prod-8 outputs from its upstream MV-mult. A new partial product is accumulated in 1 clock cycle. Fig. 8 shows the details of V-Accum.

### 3.4 ReLU and Bias Addition Unit

The activation function we use is a rectified linear unit, ReLU, which introduces the max() nonlinearity as out = max( in, 0). A set of bias vectors can be added to each PE output as shown in Fig.3. Each bias vector, biasN...biasN+7, has 8 Q(17,10) elements which are added with 8 adders to the corresponding PE output vector elements oN.. oN+7. The outputs of each adder are then compared

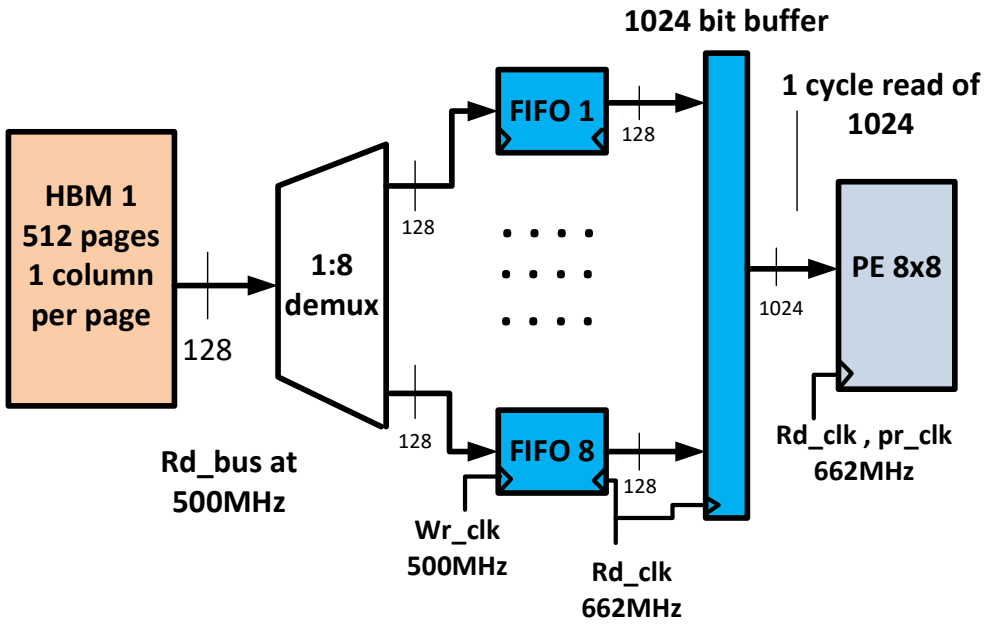


Fig. 5. Data prefetcher and on-chip buffer for HBM read accesses. One HBM is shown driving weights to its PE. The main contoller issues two read requests to column addresses Ca and Cb. Each request generates a burst of 4 128 bit transactions on DQ. The pefetcher write 8 128 bit DQ values into 8 corresponding FIFOs. A read request is then issued to all FIFOs, and their output is stored in a single 1024 bit register. This aligns the weights read-out cycle with the HBM-IN read out of the next 8 16-bit input feature values.

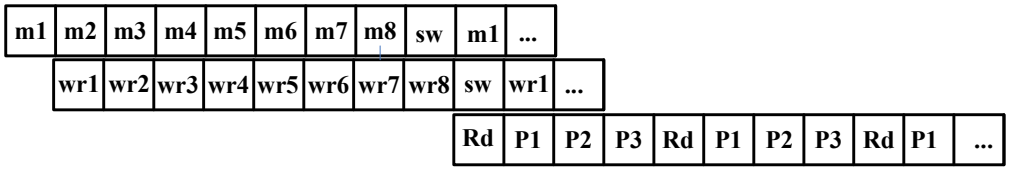


Fig. 6. DPR-BUF HBM memory read out cycles m1 to m8 in the 500MHz domain and (one of 8) FIFO write cycles wr1 to wr8. All 8 FIFOs are simultaneously read in cycle Rd and PE processing cycles P1 to P3 are in the 662 MHz domain. The 8 FIFOs are read every 4th cycle.

with 0. The combined addition and comparison are done in one clock cycle. Note that this is done only after the 512th (last) time-slot as indicated by the $t512\_en$ signal in Fig. 3. Each element is then written into a 1024 entry FIFO for streaming to the Output Feature Memory. The write clock is 100 MHz, the read clock is 150 MHz, and the FIFO implements clock-domain-crossing.

### 3.5 Main Processing Sequence

The main controller shown in Fig.3 implements the processing sequence shown in Fig. 9.

The sequence is for a 4096-1000 layer such as Alex-8 (FC8) in AlexNet or VGG-8 (FC8) in VGG16, as in Table III in [8] . Using 8x8 tiles for the weights matrix in Fig.2, the equivalent matrix of tiles

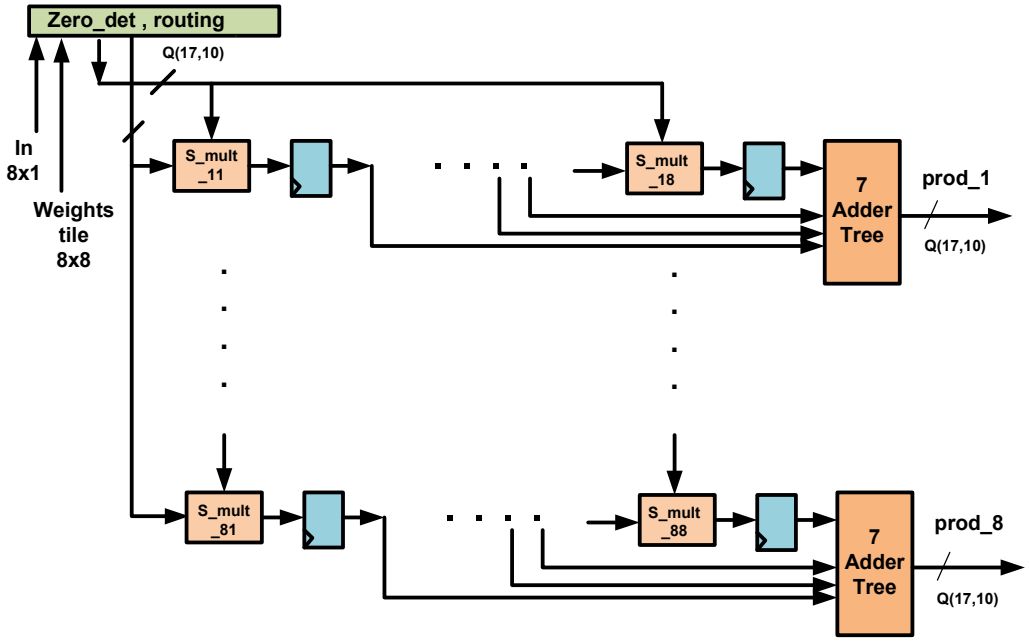


Fig. 7. MV-mult micro-architecture for 8x8 tile of weights.

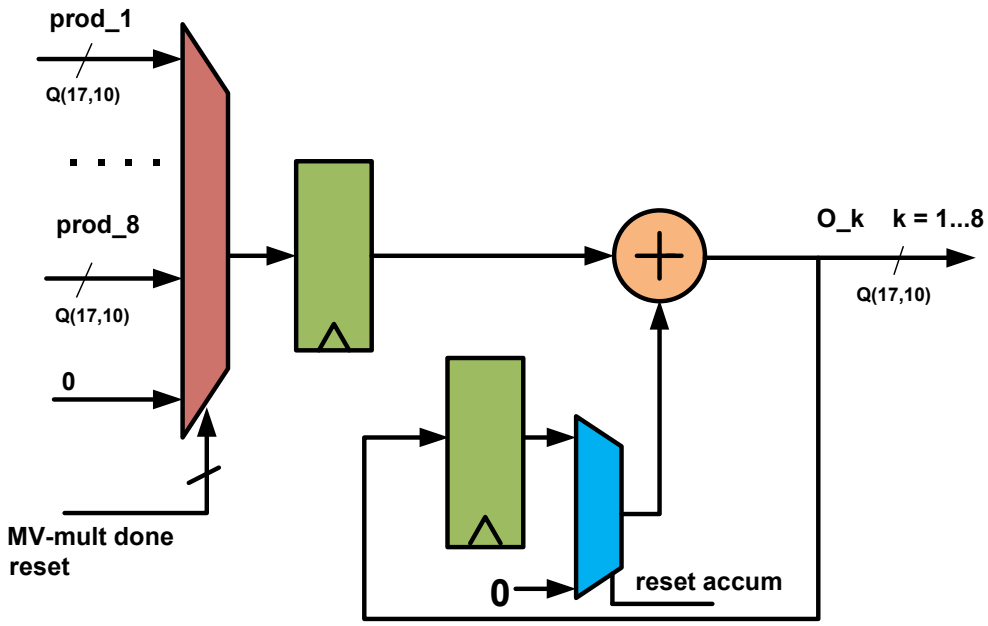


Fig. 8. Vector accumulator V-Accum datapath.

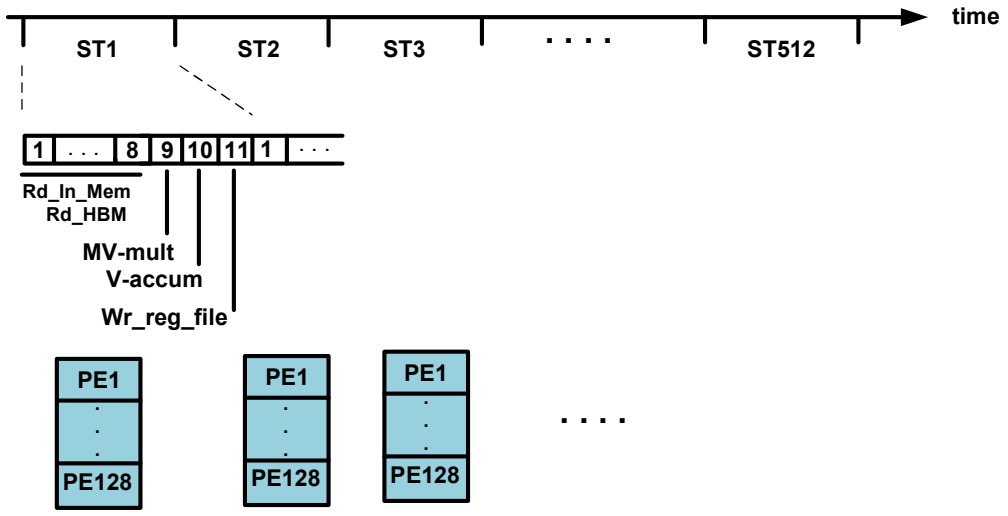


Fig. 9. Main controller sequence. All 128 PEs are processing a new 8x1 feature vector from In-MEM in each state ST1 ... ST512. In each state an 8x8 tile of weights is read from the HBM corresponding to each PE.

is 128x512. The main control sequence therefore has 512 states, ST1 to ST512 as shown. All 128 PEs are processing an input 8x1 feature vector with their corresponding tile of weights in each state. Each column is processed in sequence, and all rows in a column are processed in parallel with each row's dedicated PE. We call this a column-row-column schedule. This schedule ensures that the computing load is equally shared among all PEs. It also achieves almost optimal load balancing among all PEs since all are utilized in each control time slot. Using this schedule we read the entire input features memory only once, where each read transaction returns a 8x1 or 16x1 vector of inputs. Similarly each row's HBM weights memory is also read only once. This minimal access pattern to the memories contributes to low processing latency and minimizes power consuption. The same control sequence, ST1 to ST512, can be applied to 4096-4096 layers such as FC7 Alex-7 and FC7 VGG-7 in [8]. To maximize throughput, 512 8x8 PEs in one pass, can be used for processing in each state. Alternatively, 128 16x16 PEs can be used in two passes, for details see section "Up-Scaling to Larger FC Layers" below. For the larger layers, eg.FC6 Alex-6 , 9216-4096, the control sequence has ST1 to ST1152 or 1152 states. The number of 8x8 PEs remains at 512 for one pass. For FC6 VGG-6, 25088-4096, the control sequence has ST1 to ST3136 or 3136 states.The number of 8x8 PEs remains at 512 for one pass .

## 4 SIMULATION RESULTS

### 4.1 Simulation Setup and Comparisons to Benchmarks

The FC-Accel microarchitecture for the Alex-8/VGG16 FC layers was implemented in fixed-point Q(17,10) (data and weight) Verilog and simulated in ModelSim SE. A Python floating-point implementation of the same layer was used as reference. Pipelined and non-pipelined PEs were implemented with 662MHz and 100MHz clocking respectively. The seven adders tree in the original PE was pipelined to reduce it's critical path delay to 1.51 nsec and a 662MHz clock Non-zero values were used for all data features and for all weights. The following Table 1 summarizes the achieved processing latency, for FC8 layer inference processing, for the specified design parameters and

compares with recent comparable benchmarks. Fields without published measured data are left blank.

Table 1. Processing Latency Comparisons, FC8 layer : Unit us

| Platform | AlexNet | VGG16 |
|---|---|---|
| Versal FPGA VC1902, DPU-96AI, Batch 1, 1.33GHz, | | 158 |
| GPU (Jetson ATX) with DLA, Avg of 14023 iter, | | 180 |
| GPU (Jetson ATX) without DLA, Avg of 13744 iter, | | 120 |
| GPU (Titan X) Batch size 1, dense | 80.5 | 80.5 |
| GPU (Titan X) Batch size 64, dense | 5.9 | 5.9 |
| Arria-10 FPGA DLA, 1.2 GHz, | | 26 |
| EIE ASIC with compression,pipelined PE, 800MHz | 9.9 | 8.4 |
| this work (non-pipelined ASIC 8x8 PE, 100MHz) | 56.32 | 56.32 |
| this work (pipelined ASIC 8x8 PE, 662MHz) | 8.5 | 8.5 |

The fully connected layer FC8 in both AlexNet and in VGG16 has the same 4096-1000 dimensions. Our FC-Accel latency is based on non-zero values for all input features and all weights. The data for GPU Titan X and EIE ASIC is from [8]. The Titan X GPU (batch size 64) achieves the lowest latency of 5.9 usec but at a high cost : 3072 CUDA cores, 1 GHz clock, 28 nm CMOS process, and peak power dissipation of 250 W. We summarize a pipelined 8x8 PE FC-Accel implementation using an 662 MHz clock (45 nm PDK CMOS process) and 7 pipeline stages for the 7 adder tree in Fig.7. The non-pipelined version uses 100 MHz clocking. Using pipelining brings the worst case critical path delay to 1.51 nsec and considerably improves latency. However it increases power dissipation as shown below.

Table 2 reports the operations/sec for each major processing block in FC-Accel.

Table 2. Processing Blocks Performance

| Block | GOPS |
|---|---|
| MV-mult, non-pipelined, all 512 FC8 time slots | 1536 |
| MV-mult, pipelined, all 512 FC8 time slots | 10172 |
| V-accum , all 512 FC8 time slots | 204.8 |
| Add-bias, ReLU, final FC8 time slot | 102.4 |

The total (dynamic and leakage) power consumption in the pipelined 8x8 PE is shown in Table 3 for each processing block along with the cell counts.

Table 3. Power per processing block in pipelined 8x8 PE

| Block | Power | Cells |
|---|---|---|
| MV-mult 8x8, pipelined | 581.6 mW | 140662 |
| V-accum 8x1 | 12.3 mW | 2468 |
| Total PE | 593.9 mW | 143130 |

Table 4 compares the achieved operations/sec for the 4096-1000 FC8 layer with other comparable benchmarks (all units are in GOPS) and shows the speedups achieved by FC-Accel in ASIC PDK 45

nm technology. The FPGA implementations are from Table 17 in [15] . The TETRIS implementation is from [5].

Table 4. Comparison with ASIC and FPGA Platforms for layer FC8 Acceleration in GOPS : A = AlexNet V = VGG16

| Platform | A | V |
|---|---|---|
| EIE ASIC 45nm with compression,pipelined PE | 102 | 102 |
| TETRIS ASIC 45nm, 500MHz, | 627 | 627 |
| VC707 FPGA 28nm, 150MHz, 13.5W | 28.8 | 131.2 |
| ZC706 FPGA 28nm, 150MHz, 8.9W | 16.5 | 71.2 |
| Versal FPGA DPU-96AI, 1.33GHz | | 51.87 |
| Arria-10 FPGA DLA, 1.2GHz Memory | | 1378 |
| this work, 128 non-pipelined 8x8 PEs, 100MHz, 17W | 108 | 108 |
| this work, 128 pipelined 8x8 PEs, 662MHz, 90.1W | 1048 | 1048 |

Note that the EIE ASIC is using a 800 MHz processing clock. The TETRIS ASIC is using 16 3D engines, with 16 HMC memory stacks. TETRIS uses Hybrid Memory Cube, HMC, which is also a 3D memory stack but architecturally different from an HBM 3D memory stack. The Arria-10 DLA is implemented in a 20 nm process technology, a more advanced process than PDK 45 nm used for the FC-ACCEL. The DLA also uses 1.2 GHz clocking to its memory sub-system, versus 500 MHz clocking in this work, which allows the DLA to achieve a peak of 1378 GOPS during FC8 inference processing, [10]. Note that in this case, even with more operations per second, the FC8 inference latency, 26 usec, is still larger than FC-ACCEL's FC8 latency. Similarly, the NVIDIA Jetson ATX GPU [18] has a reported 32 Tera Ops, TOPS, Deep Learning operations (TOPS DL), but achieves an inferencing latency of only 120 usec when the VGG16 FC8 layer is profiled, see Table 1.

### 4.2 CMOS ASIC Implementation

We have implemented the Alex-8/VGG-16 FC8 layer using the CMOS ASIC PDK 45 nm standard cell library for synthesis. The Cadence RTL Compiler (RC) tool was used and the design achieved timing closure with a 100 MHz clock for the non-pipeliend PE and 662 MHz for the pipelined PE.

Table 5 summarizes timing, area, and power for the non-pipelined and pipelined FC-Accel design, with 128 8x8 pipelined (662 MHz) or non-pipelined (100 MHz) PEs.

Table 5. FC-Accel PDK45 Standard cell Implementation

| Design Technology | NCSU PDK 45 nm | |
|---|---|---|
| clk freq | non-pipelined | pipelined |
| std cell VDD | 1 V | 1V |
| Combinatorial gates | 11188035 | 13245537 |
| Sequential Cells (DFFs) | 313480 | 844936 |
| Dynamic power | 16.9 W | 89.8 W |
| Total power (Leakage,Dynamic) | 17.2 W | 90.1 W |

### 4.3 Energy Efficiency Characterization

In this section we present a comparison of GOPS/W for the proposed FC_ACCEL implementation and several other accelerator implementations with published GOPS and power values. Some are based on HBM memory while others use DDR4 arrays. For FC_ACCEL the power estimates are for the synthesized netlist from the Cadence Encounter-RC tool using PDK45 nm with 1V VDD and assumes worst case statistical switching. The units in Table 6 are in GOPS/W. Note that the Alveo U50 Versal HBM implementation does not implement AlexNet or VGG16 networks, but a Randomly Wired Neural Network (RWNN) similar to ResNet-50. Power consumption for an Alveo U50 card with Versal FPGA and HBM2 is listed at maximum of 75 W in [22].

Table 6. Comparison of GOPS/W for HBM-based or DDR4-based DNN Accelerators

| Platform | AlexNet | VGG16 | RWNN |
|---|---|---|---|
| VC707 FPGA DDR4 [15] | 2.13 | 9.72 | N/A |
| ZC706 FPGA DDR4 [15] | 1.85 | 8 | N/A |
| Alveo U50 Versal HBM [14] | N/A | N/A | 6.33 |
| FC_ACCEL, 128 PEs 16 HBMs non-pipelined | 6.35 | 6.35 | N/A |
| FC_ACCEL, 128 PEs 16 HBMs pipelined | 11.63 | 11.63 | N/A |

### 4.4 FC_Accel Complexity Analysis for FC8

The time complexity for sequential element-by-element multiplication of nxm matrix and mx1 vector is O(n*m), [19]. This is the number of scaler fixed-point multiplications and per-row partial-sums additions. Using 1 clock per scalar multiplication and per-row partial-sums addition, that is O(n*m) clock cycles. Here a scalar is a 16-bit fixed point value. Parallel matrix-vector multiplication improves on the time complexity and has three main variations : row-wise block (tile) striped matrix decomposition, column-wise block (tile) striped matrix decomposition, and checkerboard block (tile) matrix decomposition. The checkerboard method scales better then the others with the number of processors (PEs) in the 2D computational grid. Therefore it can provide the best time complexity. We provide a modification of the classical checkeboard method which uses a 2D computational grid of size (n/t)x(m/t) where t is the size of a square tile. Such a grid of PEs can require a very large number of PEs (eg. for FC8, 65,536 PEs in a 2D grid will be required for an 8x8 tile. Such a large number of hardware PEs can be very expensive in a hardware implementation). We avoid a resource expensive hardware 2D grid by using a square tile of size t (1 tile per PE) and a 1D array of n/t PEs. A column-row-column schedule, as described above, is also used to schedule the flow of all input features and weights to all PEs in the 1D array. We have O (m/t) iterations done with the 1D array. During each iteration, a PE takes K clock cycles to register inputs, perform t*t scalar multiplications and t per-row partial-sums additions, and write back the t results.

Total time complexity of the 1D array with n/t PEs, t x t tile, and a column-row-colum schedule is :

$$O((m/t) * K * d) \tag{1}$$

where m/t is the number of iterations needed to process all weights and input features, K is all processing cycles needed by a PE, and d is time (clock cycles) for a scalar multiplication or addition. In our implementation of FC-Accel for FC8, d=1, K=4 ( all 8 input features read out in parallel in 1 cycle with 3 remaining cycles for MAC and write-back), or K=11 ( each input feature read out sequentially, in 8 cycles total, as shown in the main controller sequence, with 3 remaining cycles

for MAC and write-back), and t=8. The communications between PEs and HBM and input-features memories in FC_Accel is captured in the K term.

In contrast, a 2D hardware grid for classical checkerboard, with p processors in the grid, has an overall time complexity of

$$O(m^2/p + m \log p/\sqrt(p)) \tag{2}$$

The first term accounts for all computations while the second term accounts for communications between all PEs for passing inputs and weights between PEs. While the $m^2/p$ term can be lower than our m/t, this is achieved at the prohibitive cost of a hardware 2D grid of p PEs.

The total number of operations (scalar multiplications and additions) in our FC_Accel architecture, for all n/t iterations, is

$$O(n * m + \frac{n * m}{t}) \tag{3}$$

These are the operations running in parallel in each PE, in a 1D array of m/t PEs, with the array re-used n/t times.

### 4.5 Up-Scaling to Larger FC Layers

In this section we present estimated performance with an up-scaling of the proposed micro-architecture for the larger FC6 and FC7 layers in AlexNet and VGG16. We use a 16x16 PE for these layers in order to efficiently process the larger sizes of the weight matrices. The 16x16 tile of weights reduces the number of rows and columns of matrix in Fig.2 and simplifies the processing schedule as well. Both layers have 256 matrix rows with 16x16 PEs. Since a 16x16 PE has 256 weights for its matrix-vector multiplier, this is 256 x 16 = 4096 bits of weights for each PE in each row. The HBM's page for that row can be read in 4 cycles to deliver these bits to the row's PE. The up-scaled micro-architecture has 256 16x16 PEs and 16 HBMs, each HBM supplying 16 pages of weights to its PE in order to process the inputs in a single pass. To save resources, we propose 128 16x16 PEs and 16 HBMs with 16 pages from an HBM and two passes to process all pages : 8 pages of weights in the first pass followed by 8 pages of weights in the second pass. We also use an HBM for the input memory so that 16 16 bit words can be read out in 1 cycle to provide the 16x1 input vector to each of the 16x16 PEs. The main control sequence in Fig.9 therefore reduces from 11 cycles to 7 cycles : 4 for reading the HBM's page with weights ( overlapped with 1 cycle for reading the HBM with input features), and 3 for matrix-vector multiplication, accumulation, and write back.

Fig. 10 shows the scheduling with the up-scaled micro-architecture for FC6 and FC7 when 128 16x16 PEs and 16 HBMs are used with two passes. In the horizontal (time) direction, AlexNet FC6 requires 576 (9216/16) time slots per pass, VGG16 FC6 requires 1568 (25088/16) time slots per pass, and FC7 requires 256 (4096/16) time slots for either network.

Two sets of 8 pages in each HBM are required to store all the weights for the weights matrix row's PE. The first 8 pages are used in pass 1 and the second 8 pages in pass 2. The maximum number of weights for the FC6 layer in VGG16 is 25088x4096 or 102,760,448 weights. Using 2 bytes per weight, this requires 268 MB of storage which is easily stored in the new HBM2 16 GB part (Flashbolt) from Samsung. Each page stores 16.75 (268/16) MB of weigths. The FC6 layer in AlexNet is 9216x4096 so it has less weights and can also fit in the 16 GB HBM2.

The switching between pages in an HBM is assumed to be negligible as well as the saving of the 128 PE's outputs after each pass to output memory. Using the largest layer, FC6 VGG16, each pass will require 1568 time slots; using a pipelined 16x16 PE at 662MHz and 7 cycles per time slot will therefore require 16.6 usec for a pass. Both passes will take 33.2 usec for processing the entire

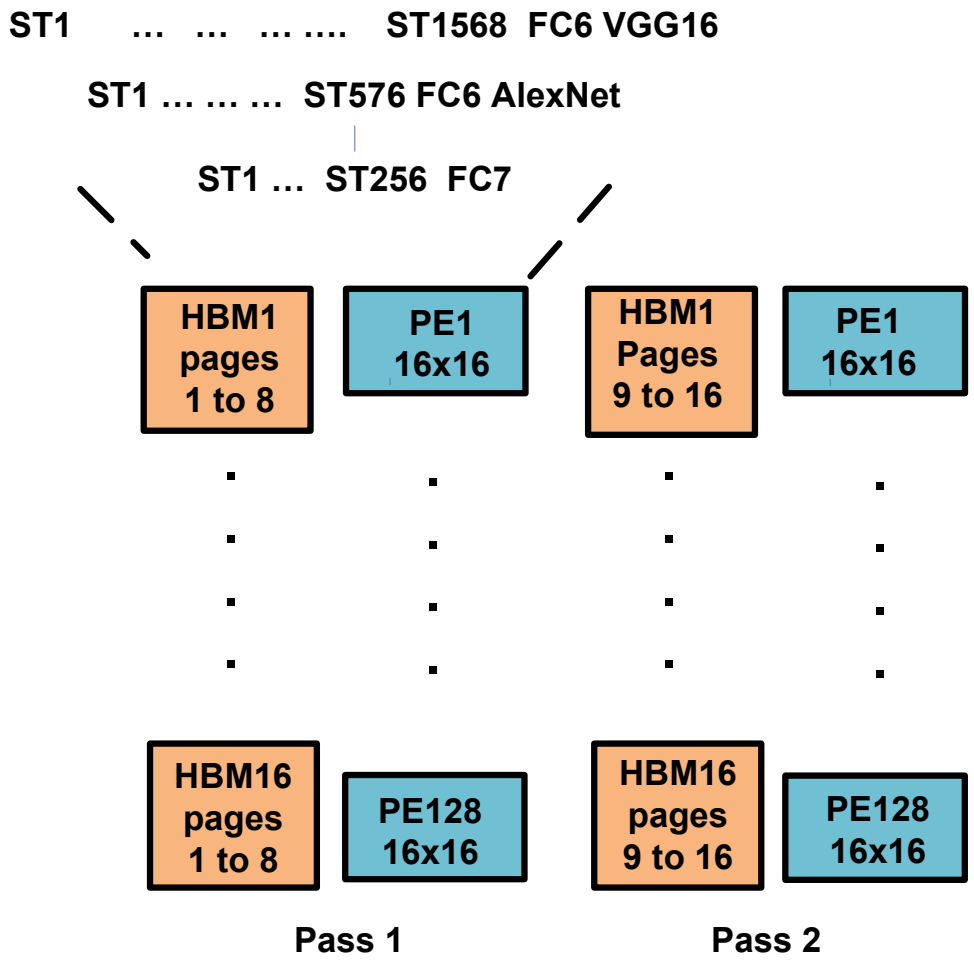


Fig. 10. Up-scaling the propsed micro-architecture for handling FC6 and FC7 layers. In each pass 16 HBMs and 128 16x16 PEs are re-used. The input and output memories are connected as in Fig.3.

layer. This is compared to 34.4 usec as reported in Table IV in [8] which uses compression and is a considerable improvement since the saving is accumulated over each forward inference.

Table 7 below summarizes the estimated total latency for FC6 and FC7 processing in both networks with the proposed up-scaled micro-architecture with pipelined 16x16 PEs. For comparison, latency from Table IV in [8] , which uses compression in each layer, is also included.

## 5 FUTURE EXTENSIONS

FC-ACCEL's 1D PE array can also be configured to perform 2D convolution with additional iterations over the 1D array depending on the size of the input features map. For example a 224x224 map (VGG16 Conv-1) will require 30 iterations or 15 macro-states with 2 iterations per macro-state. Each PE in the array can be reconfigured to perform exact 2D convolution, with an NxN input features tile, and a KxK filter kernel (weights). Defaults are 5 for N and 3 for K. The 2D convolution operation is unrolled on the PE's 8x8 core matrix multiplier, a total of 81 scalar products are

Table 7. Estimated Performance of FC6 and FC7 layers

| Layer parameter | AlexNet | VGG16 |
|---|---|---|
| this work FC6 latency | 12 usec | 33.2 usec |
| EIE [8] FC6 latency | 30.3 usec | 34.4 usec |
| this work FC7 latency | 5.41 usec | 5.41 usec |
| EIE [8] FC7 latency | 12.2 usec | 8.7 usec |

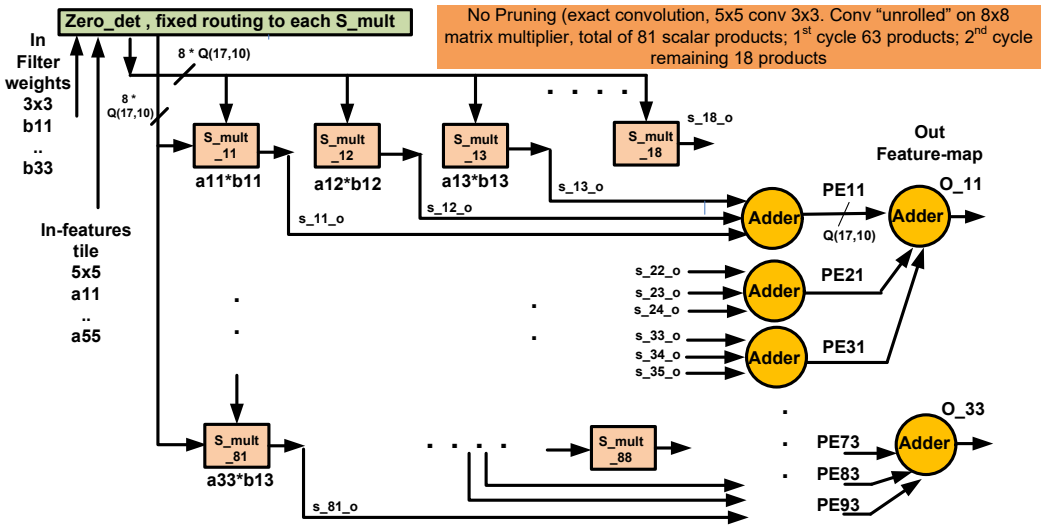


Fig. 11. PE reconfiguration to support 2D convolution of NxN input feature map (5x5 default) and KxK kernel filter (3x3 default). The same 64 8x8 matrix multipliers from Fig. 7 are reused with a different adder tree at the output.

generated in 2 iterations over the 1D array : 63 scalar products in the 1st iteration and the remaining 18 scalar products in the 2nd iteration. This is in contrast to 43 iterations required to complete the same number of scalar products in [8] and similar Eyeriss-based (non-HBM) 2D convolutional microarchitectures, [20], [3]. Fig. 11 shows the supported PE reconfiguration for 2D convolutions in the proposed HBM-based accelerator.

For example the VGG-15 CONV-1 layer, with a 224x224 input feature map, can be decomposed as 45 rows of 5x5 feature maps and 45 columns of 5x5 feature maps. Each set of 3 columns is processed in 2 iterations on the 1D 128 PEs array. All 45 columns are processed in 30 iterations. Similarly, other VGG16 Conv layers can be processed with multiple iterations over the 1D 128 PEs array.

## 6 CONCLUSION

We have discussed a novel HBM and 1D Checker-board matrix-decomposition based micro-architecture for accelerating fully connected layers in DNNs and CNNs such as FC6, FC7, and FC8 in AlexNet and VGG16. For the FC8 4096-1000 layer in AlexNet and VGG16, we achieve 108 GOPS (non-pipelined 8x8 PE) with 100 MHz at 17 W, in PDK 45nm 1V, and 1048 GOPS (pipelined 8x8 PE) with 662 MHz in the same technology. Each PE is based on a 8x8 tile of weights which can be reconfigured to 16x16. The 1D PE array can also support a 2D convolutional layer, with multiple iterations over the array, as each PE can be reconfigured for exact NxN (input features

tile) and KxK (kernel filter weights) 2D convolution. Defaults are 5x5 (NxN) and 3x3 (KxK). The achieved processing FC latency improves ( 14 % reduction in FC8 AlexNet) on recently published EIE results for the same FC8 layer without using compression. The achieved FC8 latency of 8.5 usec is smaller than the profiled NVIDIA Jetson ATX Xavier GPU (92.9 % reduction), the profiled Xilinx Versal-ACAP FPGA (94.62 % reduction), and the profiled Intel Arria-10 DLA accelerator processing the same FC8 layer (67.3 % reduction). 16 HBMs with 8 pages of weights from an HBM drive 128 8x8 PEs in one pass. The micro-architecture can easily scale up to accelerate the FC6 and FC7 layers in AlexNet and in VGG16 with the same number (16) of paged HBM memories for the weights and 128 16x16 PEs to process larger 16x16 tiles of weights. 16 pages in each of the 16 HBMs are required to support the larger layers and two passes are used, with 8 pages per pass. The estimated processing latencies for these layers is an improvement ( 60.4 % reduction in FC6 AlexNet and 3.49 % reduction in FC6 VGG16) on recently published benchmarks for the EIE [8] accelerator of the same layers.

## REFERENCES


[1] Mohamed Abdelfattah and et al. 2018. Compiler and FPGA Overlay for Neural Network Inference Acceleration. *28th International Conference on Field Programmable Logic and Applications (FPL)*. (2018), 1–9.

[2] T Chen and et al. 2014. Diannao: A smallfootprint high-throughput accelerator for ubiquitous machine-learning. *ACM Sigplan Notices* 49, 4 (2014), 269–284.

[3] Y Chen and et al. 2019. Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9, 2 (2019), 292–308.

[4] Z Du and et al. 2015. Shidiannao: Shifting vision processing closer to the sensor. *ACM SIGARCH Computer Architecture News* 43, 3 (2015), 92–104.

[5] M Gao and et al. 2017. TETRIS: Scalable and Efficient Neural Network Acceleration with 3D Memory. *Platform Lab, Standofrd Univ.* (2017), 1–25.

[6] Google. 2020. Edge TPU, Google's purpose-built ASIC designed to run inference at the edge. *https://cloud.google.com/edge-tpu/* (2020).

[7] D Hammerstrom. 1990. A VLSI architecture for high-performance, low-cost, on-chip learning. *Neural Networks, 1990., 1990 IJCNN International Joint Conference on* (1990), 537–544.

[8] Song Han and et al. 2016. EIE: Efficient Inference Engine on Compressed Deep Neural Network. *ACM/IEEE 43rd Annual International Symposium on Computer Architecture* (2016), 1–6.

[9] Li Huimin and et al. 2016. A high performance FPGA-based accelerator for large-scale convolutional neural networks. *26th International Conference on Field Programmable Logic and Applications (FPL)*. (2016), 1–9.

[10] Intel. 2017. An OpenCL (TM) Deep Learning Accelerator on Arria 10. *https://arxiv.org/abs/1701.03534* (2017).

[11] JEDEC. 2020. JESD235A,B,C HIGH BANDWIDTH MEMORY (HBM) 3D DRAM Standard. *www.jedec.org* (2020).

[12] Q Jiantao and et al. 2016. Going Deeper with Embedded FPGA Platform for Convolutional Neural Network. *ACM FPGA'16 Conference DOI: http://dx.doi.org/10.1145/2847263.2847265* (2016), 26–35.

[13] D Kim, J Kung, and et al. 2016. Neurocube: A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory. *ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)* DOI: 10.1109/ISCA.2016.41, 12 (2016), 380–392.

[14] R Kuramochi and H Nakahara. 2020. An FPGA-Based Low-Latency Accelerator for Randomly Wired Neural Networks. *IEEE 30th International Conference on Field-Programmable Logic and Applications FPL* (2020), 298–303.

[15] Sicheng Li. 2018. Towards Efficient Hardware Acceleration of Deep Neural Networks on FPGA. *Ph.D. Thesis University of Pittsburgh* (2018).

[16] Li Ning and et al. 2016. A multistage dataflow implementation of a Deep Convolutional Neural Network based on FPGA for high-speed object recognition. *IEEE Southwest Symposium on Image Analysis and Interpretation* (2016), 165–168.

[17] NVIDIA. 2018. cuSPARSE. *http://developer.nvidia.com/cusparse*. (2018).

[18] NVIDIA. 2021. Jetson AGX. *https://forums.developer.nvidia.com/t/announcing-jetson-agx-xavier-industrial/180893* (2021).

[19] M J Quinn. [n. d.]. Parallel Programming in C with MPI and OpenMP. *New York, NY, McGraw-Hill, 2004* ([n. d.]).

[20] Nitish Srivastava. 2020. Design and Generation of Efficient Hardware Accelerators for Sparse and Dense Tensor Computations. *Ph.D. Thesis Cornell University* (2020).

[21] V Sze and et al. 2017. Efficient Processing of Deep Neural Networks: A Tutorial and Survey. *IEEE Proceedings* 105, 12 (2017), 2295–2329.

[22] Xilinx. 2020. Alveo U50 Data Center Accelerator. *Data sheet DS965 (v1.7.1) August 27, 2020* (2020), 1–17.

[23] Xilinx. 2020. Versal AI Core Series. *https://www.xilinx.com/products/silicon-devices/acap/versal-ai-core.html* (2020).
[24] Q Yuran and et al. 2017. FPGA-accelerated deep convolutional neural networks for high throughput and energy efficiency. *Concurrency Computat: Pract. Exper, Wiley Online Library* 29, e3850 (2017), 1–20.