XVDPU: A High Performance CNN Accelerator on the Versal Platform Powered by the AI Engine

Xijie Jia, Yu Zhang, Guangdong Liu, Xinlin Yang, Tianyu Zhang, Jia Zheng, Dongdong Xu,
Hong Wang, Rongzhang Zheng, Satyaprakash Pareek, Lu Tian, Dongliang Xie, Hong Luo, Yi Shan
AMD
{ xijie.jia, alex.zhang, guangdong.liu, xinlin.yang, tianyu.zhang, jia.zheng, dongdong.xu,
hong.wang, rongzhang.zheng, satyaprakash.pareek, lu.tian, dongliang.xie, hong.luo, yi.shan }@amd.com

2022 32nd International Conference on Field-Programmable Logic and Applications (FPL) | 978-1-6654-7390-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/FPL57034.2022.00041

***Abstract*—The convolution neural networks (CNNs) are widely used in computer vision applications nowadays. However, the trends of higher accuracy and higher resolution generate larger networks, indicating that computation and I/O bandwidth are key bottlenecks to reach performance. The Xilinx's latest 7nm Versal ACAP platform with AI-Engine (AIE) cores can deliver up-to 8x silicon compute density at 50% the power consumption compared with the traditional FPGA solutions. In this paper, we propose XVDPU: the AIE-based int8-precision CNN accelerator on Versal chips, scaling from 16-AIE-core (C16B1) to 320-AIE-core (C64B5, Peak:109.2 TOPs) to meet computation requirements. To resolve IO bottleneck, we adopt several techniques such as multi-batch (MB), shared-weights (SHRWGT), feature-map-stationary (FMS) and long-load-weights (LLW) to improve data-reuse and reduce I/O requirements. An Arithmetic Logic Unit (ALU) design is further proposed into the accelerator which mainly performs non-convolution layers such as Depthwise-Conv layer, Pooling layer and Non-linear function layers using the same logic resources, which can better balance resource utilization, new feature support and efficiency of the whole system. We have successfully deployed more than 100 CNN models with our accelerator. Our experimental results show that the 96-AIE-core (C32B3, Peak: 32.76 TOPs) implementation can achieve 1653 FPS for ResNet50 on VCK190, which is 9.8x faster than the design on ZCU102 running at 168.5 FPS with peak 3.6 TOPs. The 256-AIE-core (C32B8, Peak: 87.36 TOPs) implementation can further achieve 4050 FPS which better leverages the computing power of Versal AIE devices. The powerful XVDPU will help enable many applications on the embedded system, such as low-latency data center, high level ADAS and complex robotics.**

***Index Terms*—CNN, FPGA, ACAP, Versal, AI Engine, Hardware Acceleration, Heterogeneous architecture, ALU engine**

I. Introduction

Machine learning using the Convolution Neural Networks (CNN) has been greatly developed in recent years. With the highly accurate predictions in image classification and object detection applications, it is applied to many scenes including auto-pilot [1], [2], medical [3] and industrial vision [4], etc. While, most of the CNN models require more computing units, memory bandwidth and a mature development environment. Based on the above reasons, researchers spend most of their time doing related works on CPU [5] and GPU [6]. However, the CPU and GPU are difficult to meet the requirements of latency and power consumption for the applications, especially for embedded scenarios. The ASIC will be a good solution [7]. But its long development period makes it hard to keep pace with the latest innovation.

With the recent development of FPGA technology especially with large on-chip storage memory and other resources, FPGAs have gained lots of popularity as specialist accelerators to improve the efficiency of CNN models. Compared with CPU and GPU, FPGA has the advantages of low latency and low power consumption. The programmability makes FPGA be more flexible than the ASIC. Engineers have developed a large number of CNN network accelerators on FPGAs. [8] proposed a hardware inference engine for sparse convolution on FPGAs. The accelerator could achieve 456-534 effective GOP/s for the modern CNNs on Xilinx ZCU102. [9] presented a novel and scalable architecture that dynamically combined the inter-layer pipelined structure and multi-layer reuse structure. The prototypes could achieve an average of 3321 GOP/s for the convolution layers for VGG16 and 2873 GOP/s for the overall ResNet50 using OctConv. Xilinx AI team presented a Deep Learning Processing Unit (DPU). It was composed of a high-performance scheduler module, a hybrid computing array module, an instruction fetch unit module and a global memory pool. Many models have been deployed on DPU including VGG, Resnet, GoogLeNet, YOLO, SSD, etc [10], [11].

Despite these advantages, the poor programmer productivity still prevents most machine learning developers to deploy the network on FPGA. Xilinx presents a new class of reconfigurable devices called the Adaptive Compute Acceleration Platform (ACAP) [12] to provide a solution for the computation and communication needs of modern applications with high performance and low power consumption. And it presents a software solution to improve the user abstraction level to develop on ACAP platform [13]. ACAP is a hybrid computing platform that integrates traditional programmable fabric, software processors and software accelerator engines (AI Engine) [14], [15]. The ACAP combines Scalar Engines, Adaptable Engines and Intelligent Engines with leading-edge memory and interfacing technologies to deliver powerful heterogeneous acceleration for any application. The ACAP is enabled by host of tools, software, libraries, IP, middleware and frameworks to enable all industry-standard design flows. The development difficulty of software engineers is reduced at the same time. It can easily deploy the related networks using the Xilinx Vitis AI tools [16].

To fully leverage the advantages of the Versal platform powered by the AIE, we propose the designed accelerator

1946-1488/22/$31.00 ©2022 IEEE
DOI 10.1109/FPL57034.2022.00041

system and the architecture. Compared with related work, the major contributions of our work are summarized as follows:

- The proposed system provides a high performance AIE-based CNN accelerator on Xilinx 7nm ACAP devices, supporting more than 100 popular CNN models.
- The designed architecture is scalable and configurable. One of the configurations with 96-AIE-core (C32B3) can provide 8x peak performance compared with Xilinx ZU9 device. Another configuration with 256-AIE-core (C32B8) can achieve 4050 fps running ResNet50.
- Several techniques, like multi-batch (MB), shared-weights (SHRWGT), feature-map-stationary (FMS) and long-load-weights (LLW), are adopted to further optimize performance and efficiency on Xilinx VCK190 board
- An Arithmetic Logic Unit (ALU) module at the Programmable Logic (PL) side is proposed to support non-convolution features with less resource utilization.

This paper is organized as follows. Sec.II introduces the top level design of the accelerator system. The details about how to use the AIE to design the CNN IP is discussed in Sec.III. The optimization strategies are analyzed in sec.IV. Sec.V describes the new ALU engine to support non-convolution operations. Sec.VI presents the performance and resources. The paper is concluded in sec.VII.

II. System Architecture

The proposed design is a configurable deep-learning processing unit based on the Xilinx Versal ACAP devices powered by the AI Engines. The designed accelerator is optimized for convolution neural networks, supporting the famous models like VGG, ResNet, YOLO, SSD, MobileNets and others.

The proposed system makes full use of the advantages of the Versal ACAP heterogeneous platform. The top level block diagram of the proposed design is shown in the Fig.1.

The AIE cores are the key roles to process the convolution calculations with high performance. The AI Engine core could perform 128x int8 MACs (Multiply-Accumulate) per cycle at 1.33GHz. The other operations are mapped on the PL (Programmable Logic, traditional FPGA parts) side for the on-chip scheduling and MISC (non-Conv) calculations. The feature-map data for each batches and the weights data shared among batches are buffered on-chip via the data-mover. The share-weights design achieves higher data-reuse and reduces the DDR IO bandwidth requirements. The other non-Conv operations are aggregated into a general engine called ALU, to simplify both the hardware and software design and reduce the PL resource utilization.

The host control is running the Xilinx Vitis AI in the PS (Processing System, ARM CPU) side to manage the runtime. The Vitis AI is also in charge of the training, pruning, quantization and compiling of the CNN models offline.

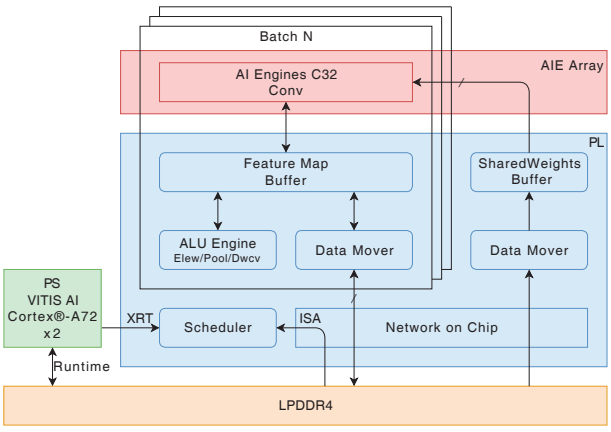

Fig. 1. Top level block diagram of the proposed design

III. Design of AI Engine

A. Function Overview

We use the AIE array [17] to do the convolution and some of the nonlinear functions associated with the convolution, including ReLU, Leaky-ReLU, hard-sigmoid and hard-swish. Each AIE core computes a sub-volume of the output feature-map (OFM) in 1 iteration. An array of AIE cores forms the AIE graph. The graph could compute multiple sub-volumes in 1 iteration. After multiple iterations a full layer of OFM is computed. The size of the sub-volume is controlled by parameters at run-time in order to achieve optimal efficiency. The parameters determining sub-volume size includes ICG (determines input channels), OCG (determines output channels), OWG (determines output columns). These parameters are calculated offline according to size of layers in the network model. At the graph level we have 3 configurations of the graph, called C16, C32 and C64, which have 16, 32 and 64 AIE cores correspondingly. Each configuration could be implemented with multi-batch, e.g. C16B1, C32B3, C64B5 according to the chip resource and performance requirement of the task model. For the multi-batch design, the multiple batches share the same set of weights (WGT) data.

Our design goal is to make accelerator as scalable as possible to be adaptive to different CNN models. The scalability has 3 levels: core level, basic graph level and batch level.

B. Core Level Design

Fig.2 shows the MAC operation in one AIE core. The basic MAC engine of the AIE core does 2*8*8 matrix-matrix multiplication in 1 cycle, the result is a 2*8 matrix. To achieve convolution operation, we map the 2*8 matrix as 2 rows and 8 channels of input feature-map (IFM), and map the 8*8 matrix as 8 input channels (IC) and 8 output channels (OC) of WGT. Therefore the result 2*8 matrix is mapped as 2 rows and 8 channels of OFM.

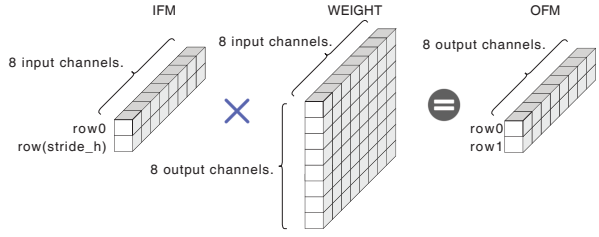

Fig. 2. The operation in one cycle of AI Engine MAC

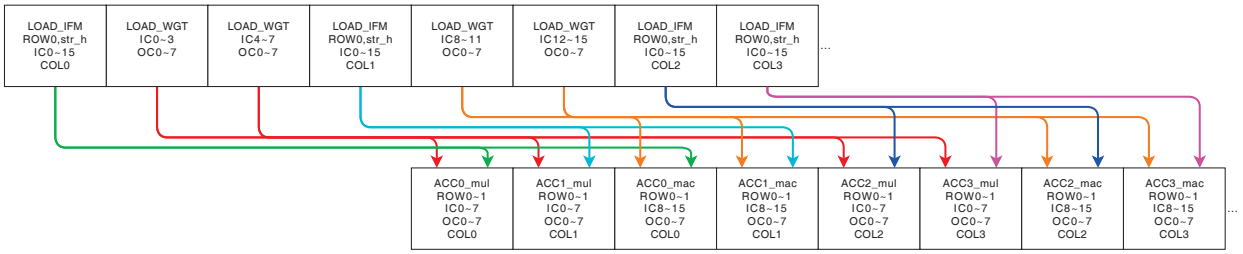

Fig. 3. The profiling of AI Engine base pipeline

The data-path of AIE is pipelined, and one of the schematics is shown in Fig 3. Each AIE core has LOAD engine to fetch data from AIE local memory to AIE MAC, which can fetch 256 bits in each cycle. To balance the fetch-data-time and the MAC-consuming-data-time, there is a minimum data size limitation for the MAC, which is called the computation granularity of the AIE core. This is the base pipeline. After the design space exploration, we set the IFM data size of the base pipeline to be 2 row * 16 IC * 4 column = 128 Bytes (int8-precision), and the WGT data size to be 16 IC * 8 OC = 128 Bytes. Then the result OFM data size is 2 row * 4 column * 8 OC = 64 bytes. Now the cycles of fetching is (128 Bytes + 128 Bytes)/256 bits = 8. And the cycles for MAC operations is 16 IC /8 * 4 column = 8. The data fetching and the computation of the base pipeline is then well balanced.

In one iteration of AIE computation, the base pipeline is looped over multiple dimensions including kernel window, IC, OC and output width. After loop on IC dimension, partial result of OFM of this iteration is produced. Selective non-linear function is done at this point.

C. Graph Level Design

By placing multiple AIE cores in different dimensions we can get the AIE graph. We have 3 basic graphs: C16, C32 and C64, indicating how many AIE cores are implemented. The dimension factors of the 3 graphs are listed in Table I. Since the computation granularity of one AIE core is 2 output rows * 8 output columns * 16 IC * 8 OC, the granularity of the graphs are listed in Table II. In the IC dimension we placed 2 AIE cores. In order to accumulate the IC of the feature-map, the cascade stream between the 2 cores are utilized. The topology of C16 is shown in Fig.4 as an example. As shown in Fig.4,

TABLE I
DIMENSION FACTORS COMPARISON OF THE DIFFERENT GRAPHS

Dimension	C16	C32	C64
Output Row	4	4	4
Input Channel	2	2	2
Output Channel	2	4	8

TABLE II
GRANULARITY COMPARISON OF THE DIFFERENT GRAPHS

Dimension	C16	C32	C64
Output Row Parallel	8	8	8
Output Column Parallel	4	4	4
Input Channel Parallel	32	32	32
Output Channel Parallel	16	32	64

each WGT stream(blue) broadcasts to 4 cores since our space factor in output row dimension is 4, each IFM stream(red) broadcasts to 2 cores since our space factor in OC dimension is 2. The C32 and C64 graphs have more AIE cores on OC dimension based on C16 graph so that the IFM streams will broadcast to more cores.

D. Multi-Batch (MB) Level Design

Each graph could be implemented with multiple batches according to the chip resources and performance requirement. Thus the multiple input images are processed synchronously. The WGT data could be shared among multiple batches, so that only one batch of WGT data is buffered. The IO bandwidth and resource utilization are reduced. The Fig.5 shows a design of C16 batch 2 graph, to be compared with the batch 1 graph in Fig.4. In the 2-batch graph the number of input WGT streams(blue) are not increased, while each WGT stream broadcast to more cores. The MB design on the PL side will be further discussed in sec.IV-A.

E. Layer Mapping Exploration

For a given CNN layer with kernel window size on width (KW) and height (KH) dimension, and stride in width (SW) and height (SH) dimension, we will explore the best parameters (ICG, OCG, OWG) offline for AIE core to determine the sub-volume size in order to get optimal efficiency. The cycles of AIE computation for 1 iteration is a function:

$f_{calc\_cycles}(KW, KH, ICG, OCG, OWG)$

Different code overheads are taken into considerations when we formulate this function like loop overheads, non-linear operation overhead, input data acquire overhead, etc. The overheads depends on the code design and compilation result of AIE core. The theory cycles is a function without overhead:

$f_{theory\_cycles}(KW, KH, ICG, OCG, OWG)$

It is formulated as total required MAC operations divided by computation parallelism of AIE core. Then the IO cycles of IFM data, WGT data and OFM data are as following, which are formulated as IO data size divided by IO stream bandwidth:

$f_{ifm\_io}(KW, KH, SW, SH, ICG, OWG)$
$f_{wgt\_io}(KW, KH, ICG, OCG)$
$f_{ofm\_io}(OCG, OWG)$

Since the iterations are also pipelined and IFM and WGT stream can work in parallel, we can estimate the real execution cycles (f_{real_cycles}) of 1 iteration as the greatest number among f_{calc_cycles}, f_{ifm_io}, f_{wgt_io}, f_{ofm_io}. Hence the efficiency would be $f_{theory\_cycles}/f_{real\_cycles}$. The problem is

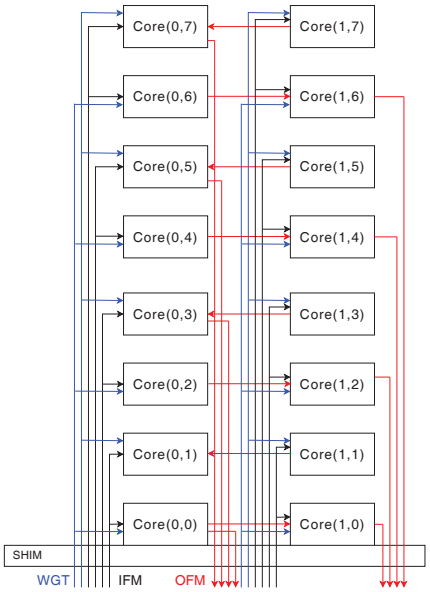

Fig. 4. The topology of the AIE C16B1 Graph

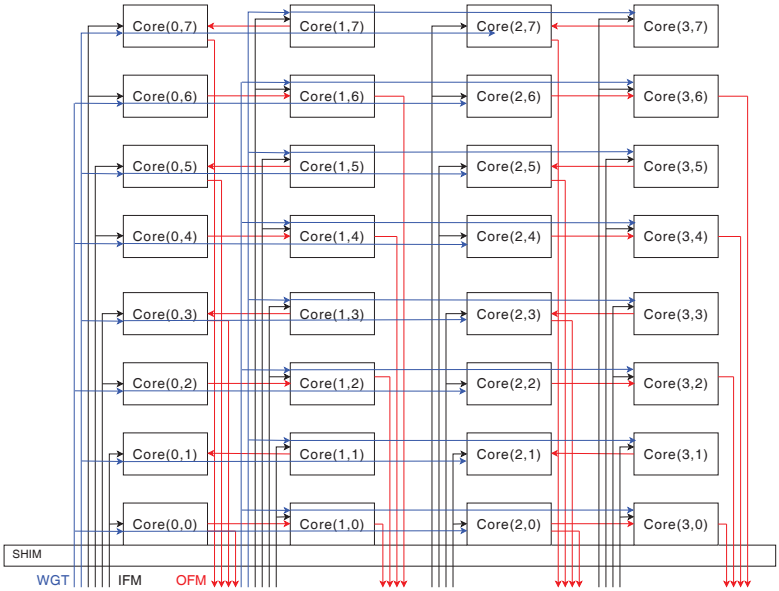

Fig. 5. The topology of the AIE C16B2 Graph. The WGT data are shared among batches

to search for a point in a 3 dimension space. The data memory size of AIE core is limited, so we have a constraint that maximum total size of IFM, WGT and OFM is fixed. This constraint makes our design space smaller so that we can calculate efficiency for every point in the space and get the best one. When we get the size of the sub-volume for 1 iteration we also get the total iteration number for the whole layer.

F. Cooperation with PL

We place IFM and WGT buffers at PL side. The AIE array is connected with PL side via AXI-stream interfaces. The streams include data streams (IFM, WGT, OFM) and control streams. The working flow of AIE cores is controlled by PL via control packets on the control streams. At the start of the processing of one layer, PL side will send control packets including layer parameters needed by AIE cores for computation. PL side is also responsible for feeding IFM and WGT to AIE array and receiving OFM from AIE array.

IV. Data Flow Optimizations

The proposed design is implemented on Xilinx VCK190 evaluation board. We adopt several techniques to do optimizations and trade-off to resolve IO bottleneck.

A. Share-Weights Data-Mover

The data-mover engines illustrated in Fig.1 are responsible for transmitting data between DDR and on-chip ram via AXI4 interfaces, including feature-map, weights and bias. In the traditional designs, one data-mover port is used for transmitting all types of data, resulting a lot of AXI4 interfaces in the multi-compute-unit design, and low DDR IO efficiency. Inspired by that different image batches require the same weights, the solution called shared-weights (SHRWGT) is adopted on the multi-batch (MB) design: an additional load-weights engine and its global weights memory are shared among batches, which reduce requirement of both DDR bandwidth, AIE shim bandwidth and on-chip RAM resources.

The amounts of AXI4 interfaces are configurable. For feature-map ports, whose data width is 128 bits, could be set from 1, 2, 4, to 8 for each batch. Typically we use 2 ports per batch, applying for 128 bits * 2 ports * 333 MHz = 10.6 GB/s. For shared-weights ports, whose data width is 512 bits, could be set as 2 and 4. Typically we select 4 ports, applying for 512 bits * 4 ports * 333 MHz = 85.2 GB/s. Considering that VCK190 board could only provide 68.3 GB/s DDR bandwidth, SHRWGT solution alleviates the insufficient bandwidth issue.

B. Feature-Map-Stationary (FMS [18])

Typically, data flow of feature-map between layers have to interact through DDR IO. DDR space is large enough to store various sizes of feature-map. Reading and writing of DDR are flexible through AXI4 interfaces. But the latency of DDR IO is large in data flow, which will lead to performance loss.

Therefore, we adopt FMS strategy to keep feature-map stay at on-chip RAM during its life cycle. The rich UltraRAM resources on Versal chip make it achievable. In this way, not only bandwidth pressure of DDR IO is reduced, but also data acquisition time is shortened. Certainly, not all styles of feature-maps can achieve FMS. The most basic requirement is that feature-map can fit on the on-chip RAM. Also, it is necessary to ensure that there is enough RAM space left for the remaining layers to store the intermediate results during the period that it occupies the on-chip RAM.

The Fig.6 illustrated an application example of FMS on a model called MLPerf_resnet50_v1.5_tf [19]. Except for loading-IFM in the first layer and saving-OFM in the last layer, there are no more DDR IO access for the feature-map. The details will be discussed in sec:VI-B.

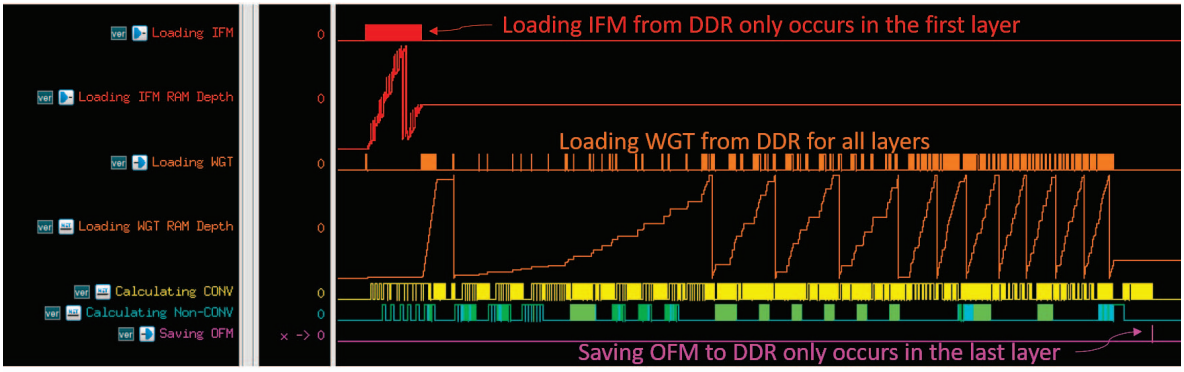

Fig. 6. Waveform of MLPerf_resnet50_v1.5_tf with Feature-Map-Stationary Strategy. The red line: loading IFM only occurs in the first layer. The magenta line: saving OFM only occurs in the last layer. The FMS Strategy buffers all intermediate feature-map on the on-chp RAM. For the inner-layers, only the orange line: loading wgt is processing. The DDR IO is well reduced.

C. Long-Load-Weights (LLW)

With the help of FMS strategy, the DDR IO requirements for loading and saving images could be reduced a lot, as shown in Fig.6. The figure also shows that only the weights data of each layers should be loaded into the on-chip RAM from DDR. In some large layers, the DDR IO for weights data is the main bottleneck. To optimize this point, the LLW design is adopted.

The weights on-chip buffer is divided into 16 banks in our design. Before the LLW is adopted, the WGT data in DDR is arranged linearly, in sequence of IC, KW, KH and OC dimension. One load instruction could only transmit for one OC into one bank, via 128 bit data-width. Then lots of load instructions are required. In the LLW design, the weights data arrangement in the DDR is shuffled, as shown in Fig.7. The OC dimension is divided into 4 groups, so we select 4 weights ports. The IC dimension in each OC group is interleaved per 128 bits, then the data-width of each weights port could be increased to 512 bits, as mentioned in sec.IV-A. Thus, one long load instruction could transmit for 4*N OC into four banks with continuously DDR access. The DDR IO efficiency is improved a lot. The LLW strategy along with the larger DDR IO bandwidth mentioned in sec.IV-A, could help reduce the DDR access bottleneck.

V. Design of ALU Engine

The CNN models are evolving rapidly with the new non-conv operations. Besides convolution layer (Conv), the state-of-the-art CNN models are generally composed of the following operations: Depth-wise Convolution (DWconv), Max-Pooing and Average-Pooling, Activation function (ReLU, LeakyReLU, PReLU, ReLU6, etc.), Batch Normalization (BN), Element-wise addition (Elew_add), Element-wise multiply (Elew_mult), Full-connected layer (FC) and so on.

We have discussed how we map the Conv operation onto AIE in section III. For the non-Conv operations, before the ALU engine is proposed, we considered deploying the operations to CPU or designing a dedicated hardware accelerate engine for each operation, such as DWConv engine, Element-wise engine and Pooling Engine. But these two methods have obvious shortcomings. 1) Deploying operation to CPU brings larger latency. 2) Designing dedicated calculation engines consumes too many on-chip resources, indicating high power consumption and low energy efficiency. Generally, the non-Conv operations will not be triggered at the same time due to the inter-layer data dependency. This enables us to design a multi-function hardware accelerator using shared hardware resources on PL side which is called the ALU engine, to perform non-Conv operations. The combined design of AIE for Conv and ALU for non-Conv can better balance the resource utilization, new feature support and efficiency of the whole system.

A. Operation Modes

The ALU engine supports 5 basic operation modes listed in Table III: ALU_Add, ALU_Mult, ALU_Comp, ALU_Macc and ALU_PReLU. Based on these basic modes, the ALU engine can support non-Conv operations through the adjustment of instructions and weight/bias parameters. The ALU engine also support wide parameter range, such as the kernel size ranging from 1x1 to 256x256 for Pooling and DWConv

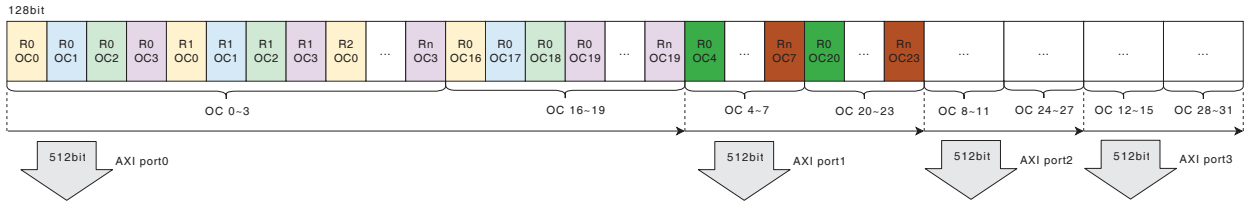

Fig. 7. The shuffled WGT data arrangement in DDR for Long-Load-Weights. Assuming there are 32 output channels of weights, then OC0-3 and OC16-19 are placed together. Every 128 bits of OC0-3 are interleaved to be 512 bits of R0-Rn. The AXI4 interface can access them in one cycle

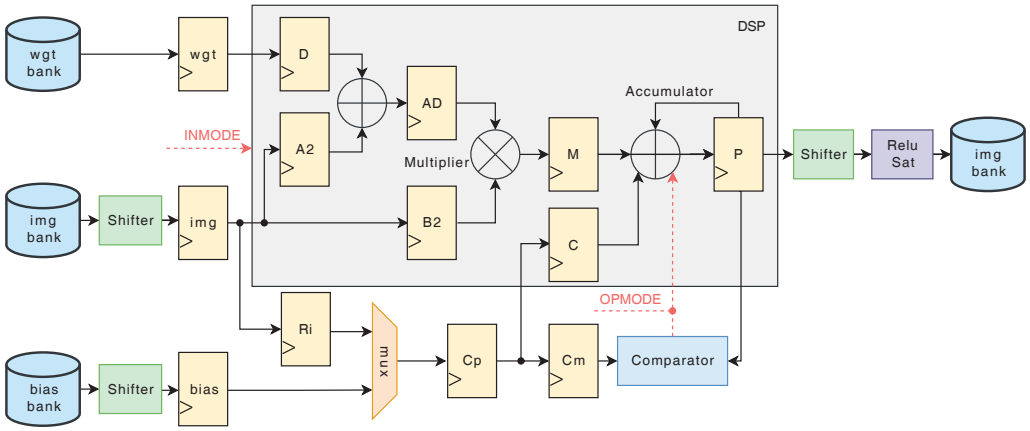

Fig. 8. The function diagram of ALU PE

TABLE III
THE BASIC OPERATION MODES AND THEIR SUPPORTED OPERATION LAYERS OF ALU ENGINE

Basic Op Mode	Supported Operations
ALU_Add	Elew_add
ALU_Mult	Elew_mult
ALU_Comp	Max-Pooling
ALU_Macc	DWConv, Avg-Pooling, Hard-Sigmoid, Hard-Swish*
ALU_PReLU	LeakyReLU, PReLU

* based on ALU_Macc and ALU_Mult

operations. Thus, many popular CNN models can be deployed in our hardware accelerator with the help of the ALU engine.

For example, when supporting EfficientNet network [20], the activation function sigmoid $\sigma(\bullet)$ in Eq.(1) [21] shall be compute but it's still difficult to map directly to PL side. Firstly Eq.(1) is approximated to Eq.(2). But the division in Eq.(2) is also not friendly enough. Next step to Eq.(3) by replacing the division with shift operation which is easy to be implemented. Therefore, the calculation of Eq.(3) is mapped with ALU_Macc mode. For most of cases, the IFM/WGT/Bias for ALU engine are all int8-precision. When dealing with Eq.(3), the WGT/Bias shall be int16-precision constants, which is also supported well by ALU engine.

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{1}$$

$$h\sigma(x) = \begin{cases} 0, & x < -3 \\ \frac{x+3}{6}, & -3 \le x \le 3 \\ 1, & x > 3 \end{cases} \tag{2}$$

$$hf\sigma(x) = \begin{cases} 0, & x < -3 \\ \frac{(x+3)*2731}{2^{14}}, & -3 \le x \le 3 \\ 1, & x > 3 \end{cases} \tag{3}$$

B. Implementation of ALU PE

The function diagram of the ALU Processing Element (PE) is illustrated in Fig.8. The operation modes are processed by the following modules:

- **Shifter**: Used to support different fixed-point positions set by the quantization steps, both for the IFM/Bias and the OFM. The shift bits and direction are acquired via the instruction.
- **Multiplexer**: Each operation mode has its dedicated internal circuit data-path. The multiplexer selects the corresponding circuit path and data source according to the instruction.
- **Multiplier**: The multiplier performs the multiplication of two fixed-point data, which is mapped to the DSP Slice to save the valuable CLB resource.
- **Accumulator**: The 3-input post-adder summaries the result of the multiplier (register M), the addend (register C) and the augend (register P) to realize the the accumulation. Each of the these inputs could be turned-off as 0 according to the operation modes.
- **Comparator**: The comparator compares two fixed-point data and outputs 1-bit *opmode* signal to control the inputs of accumulator for Max-Pooling layers.
- **Relu/Saturation** The final results are performed by the activation function, including ReLU and ReLU6, and are saturated and truncated to int8 type for the next layer.

The Versal devices have many dedicated low-power DSPs combining high speed with small size while retaining system design flexibility. The DSP Engine in the Versal architecture is defined using the DSP58 primitive [22], which is the sixth version of the Xilinx DSP, and fully backwards compatible with the previous DSP48E2 version [23]. As shown in the Fig.8, an ALU PE is well mapped on a single DSP Slice. The WGT data is connected to the input D. The Bias data and IFM data share the input C through multiplexer. The IFM data is also connected to input A and B. The pre-adder is mapped as a multiplexer by setting the *inmode* to perform either D*B2 or A2*B2. The *opmode* is used to dynamically control the accumulator (post-adder) behaviour.

C. Computing Parallelism

The ALU engine is an array composed of multiple ALU PEs running at 333 MHz. The number of ALU PEs can be

TABLE IV
CONFIGURABLE PARAMETERS OF THE PROPOSED SYSTEM

Parameter	Setup	Description
CPB_N	32	Amounts of AIE cores per batch, support 16, 32 and 64.
BATCH_N	3 and 8	batch size, For CPB_N=16,32, BATCH_N can be 1 to 8. For CPB_N=64, BATCH_N can be 1 to 5.
ALU_N	128	Parallelism of ALU PE, support 16, 32, 64, 128.
HP_IMG_N	2	Amounts of AXI4 128-bit interfaces per batch for feature map, support 1, 2, 4, 8.
HP_WGT_N	4	Amounts of AXI4 512-bit interfaces for weights, support 2, 4.
PL_FREQ	333MHz	Frequency of PL side, support up-to 333 MHz.
AIE_FREQ	1.333GHz	Frequency of AI Engine side, support up-to 1.333 GHz.

TABLE V
COMPARISON OF RESOURCE UTILIZATION

	ZCU102 B4096CU3	Ours C32B3	Ours C32B8
FFs	315k	266k	656k
LUTs	160k	205k	516k
36kb BlockRAM	771	0	870
288kb UltraRAM	0	332	439
DSP Slice	1686	407	1077
AI Engine	0	96	256
Peak Performance	3.69TOPS	32.76TOPs	87.36TOPs

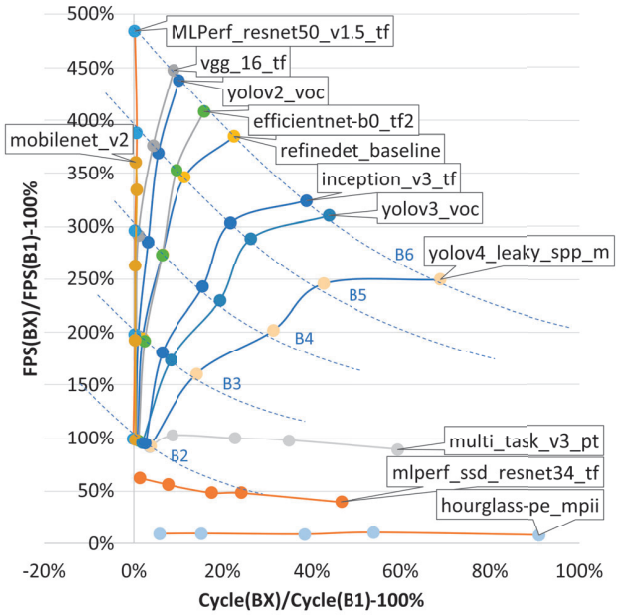

Fig. 9. Performance% vs Cycle% on C32 Multi-Batch ranging from C32B1 (32-core) to C32B6 (192-core). The performance limitation curves of each batch are shown as the dashed line.

configured according to the requirements of computing parallelism and hardware resources. The computing parallellism of feature map is scalable along input row dimension and channel dimension. In our design, the calculation parallelism of ALU engine is 128, namely there are 128 ALU PEs, implementing with 128 DSP slices per batch.

VI. EVALUATIONS

The proposed system was implemented on the Xilinx VCK190 evaluation board [24], which features Xilinx Versal ACAP XCVC1902 device. The system is built with the Xilinx Vitis 2021.1 tool-chain environment. The following section presents the evaluation results.

A. Implementation Setup

The table IV summaries the configurable parameters of the proposed scalable system and the selected values in this evaluation. The resource utilization of the implementation is shown in Table V. We select the Xilinx DPUCZDX8G B4096-3-Core design [10] implemented on the Xilinx ZCU102 as comparison. It can be seen that AIE replaces a large number of DSP slices and provides much higher compute power. And now the on-chip DSP slices are available for other features, like ALU engines.

B. Performance on the ModelZoo

The proposed system supports a wide range of int8-precision CNN models, like classification, segmentation, face detection, etc. Table VI lists the performance of some typical networks in Vitis AI Library ModelZoo [19] which includes more than 100 popular CNN models. The workload, which is the amount of multiplication and accumulation of convolution in the model, is used to measure the computational complexity of different models to uniformly describe different types of network structures. From the frames per second (fps) of end2end (E2E), the performance of our design is significantly improved compared to the ZCU102 design.

Our C32B3 (96-core) design achieves 8.8x (32.76 TOPs vs 3.69 TOPs) peak performance compared with DPUCZDX8G B4096-3-Core design. While the VCK190 evaluation board only provides 3.4x (68.3 GB/s vs 20 GB/s) DDR IO bandwidth compared with ZCU102 board. This mismatch is insufficient to IO requirement of some large models, especially for our multi-batch (MB) design. The Fig.9 illustrates the relationship of the performance vs execute cycles on our MB design, ranging from C32B1 (32-core) to C32B6 (192-core). For many models, like yolov3_voc that couldn't achieve FMS, will have to run with more cycles at the larger batch sizes, which is limited by the DDR IO bandwidth. Then their performance drop along the limitation curve shown as the dashed line. Therefore, their acceleration ratios vs ZCU102 are dropped as well. Besides, there are some models that didn't reach the limitation curve, like mobilenet_v2, for that CPU processing pre- and post-processing is the bottleneck.

The evaluation indicates that proposed accelerator not only provides more powerful computing power but also optimizes efficiency. Taking the waveform of MLPerf_resnet50_v1.5_tf as an example as shown in Fig.6, the larger on-chip RAM enables the FMS strategy mentioned in sec.IV-B. The process of loading IFM only occurs in the early stage of model calculation. All intermediate data is cached in the on-chip RAM reducing the requirement for DDR IO bandwidth and improving the computational efficiency. The higher performance is

TABLE VI
PERFORMANCE COMPARISON ON VARIOUS TYPES OF CNN MODELS [19]

Category	Model Name	Workload GOPs	ZCU102 B4096CU3 latency/ms 1 thread	ZCU102 B4096CU3 fps 6 threads	Ours C32B3 96-core latency/ms 1 thread	Ours C32B3 96-core fps 6 threads	FPS Improvement Ratio
classification	MLPerf_resnet50_v1.5_tf	8.18	14.04	168.5	3.15	1653.50	9.81
	vgg_16_tf	30.947	49.75	41.1	9.27	375.062	9.13
	inception_v3_tf	11.451	16.99	135.9	6.98	610.357	4.49
	mobilenet_v2 *	0.602	4	734	2.27	2970.11	4.05
	efficientnet-b0_tf2	0.36	/	/	3.36	994.15	/
ssd	mlperf_ssd_resnet34_tf *	432.884	559.75	7.1	284.17	22.8719	3.22
yolov2	yolov2_voc	34.063	37.88	70	10.68	459.498	6.56
yolov3	yolov3_voc	65.428	78.5	33.5	19.21	199.672	5.96
yolov4	yolov4_leaky_spp_m	60.112	75.45	33.8	25.3	166.621	4.93
facedetect	densebox_640_360 *	1.107	4.78	812.9	5.94	1048.63	1.29
hourglass	hourglass-pe_mpii *	11.669	54.76	73.5	41.82	133.654	1.82
refinedet	refinedet_baseline	123.898	117.37	24.7	25.05	149.68	6.06
reid	reid	0.968	2.85	692.7	1.076	5027.15	7.26
multitaskv3	multi_task_v3_pt	25.035	60.01	60.8	42.26	183.9	3.02
rcan	rcan_pruned_tf	86.946	/	/	63.15	60.8645	/

* indicates Performance limited by CPU bottleneck.

TABLE VII
COMPARISON OF MLPERF_RESNET50_V1.5_TF

	CPU	GPU	GPU	GPU	ZCU102 B4096CU3	Ours C32B3	Ours C32B8
Platform	Intel(R) Xeon(R) Gold 6136	nVidia Tesla V100-PCIE-16GB	nVidia Jetson AGX Xavier	nVidia Jetson AGX Xavier	Xilinx ZU+ MPSoC ZU9	Xilinx ACAP VC1902	Xilinx ACAP VC1902
Frequency	3.00GHz	3.00GHz	2.26GHz	2.26GHz	300MHz	PL@333MHz AIE@1.333GHz	PL@300MHz AIE@1.333GHz
Power	150W (TDP)	250W (TDP)	30W (TDP)	30W (TDP)	25W (TDP) 8.6W (Idle) 16.3W (Average)	125W (TDP) 22.96W (Idle) 36.1W (Average)	125W (TDP) 30.98W (Idle) 62.8W (Average)
Precision	float32	float32	float32	int8	int8	int8	int8
Latency (batch_size)	29.8ms (1)	2.38ms (1)	8.68ms (1)	1.79ms (1)	14.0ms (1)	1.80ms (1)	1.91ms (1)
FPS (batch_size)	97.1 (32)	1552 (32)	170 (32)	1346 (32)	168.5 (3)	1653.5 (3)	4050 (8)

achieved. Thus, our proposed system with C32B3 (96-core) achieves 9.81x E2E performance improvement than that on ZCU102, which is higher than the theoretically 8.8x peak performance improvement. When the batch size increases, the FMS strategy keeps the model's cycles be stable as shown in Fig.9. Furthermore, we have explored the C32B8 (256-core) configuration (the maximum batch size are limited by the on-chip RAM resources). Its performance reaches up-to 4050 FPS (Table VII), which achieves 24x performance acceleration compared with DPUCZDX8G on ZCU102 and 1.64x with DPUCVDX8G-C32B6 on VCK190 [25].

VII. CONCLUSION

In this paper, a high performance CNN accelerator on the Versal platform powered by the AIE is proposed. The Heterogeneous computing system, AI Engine + PL fabric + PS CPU, leverages the advantages of 7nm Versal ACAP platform. The C32B3 (96-core) AIE design provides 8.8x times peak performance compared to the previous 16nm ZU9 MPSoc device using the DSP slice. While implementing to Xilinx VCK190 board, to resolve its insufficient DDR bandwidth (68.3GB/s), several optimizations, like MB, SHRWGT, FMS and LLW are adopted to improve IO efficiency and reduce on-chip resource utilization. Furthermore, the newly designed ALU engine could map kinds of non-conv operations onto the single DSP slice, which provides the functional generality with compact resource utilization. The evaluations of the ModelZoo test on VCK190 board show that, the proposed design supports more than 100 popular CNN networks, and achieves a higher performance for the CNN inference task. The C32B8 (256-AIE-core) implementation can further achieve 4050 FPS for the ResNet50 model. In the future, we will continue to support more models and to further improve the inference efficiency and performance.

REFERENCES

[1] H. Gao, B. Cheng, J. Wang, K. Li, J. Zhao, and D. Li, "Object classification using cnn-based fusion of vision and lidar in autonomous vehicle environment," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 9, pp. 4224–4231, 2018.

[2] K. Pranav and J. Manikandan, "Design and evaluation of a real-time pedestrian detection system for autonomous vehicles," in *2020 Zooming Innovation in Consumer Technologies Conference (ZINC)*. IEEE, 2020, pp. 155–159.

[3] S. Sudha, K. Jayanthi, C. Rajasekaran, and T. Sunder, "Segmentation of roi in medical images using cnn-a comparative study," in *TENCON 2019-2019 IEEE Region 10 Conference (TENCON)*. IEEE, 2019, pp. 767–771.

[4] T. Gulde, D. Ludl, and C. Curio, "Ropose: Cnn-based 2d pose estimation of industrial robots," in *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2018, pp. 463–470.

[5] M. D. Putro, D.-L. Nguyen, and K.-H. Jo, "Fast eye detector using cpu based lightweight convolutional neural network," in *2020 20th*

International Conference on Control, Automation and Systems (ICCAS). IEEE, 2020, pp. 12–16.
[6] S. Lym, D. Lee, M. O'Connor, N. Chatterjee, and M. Erez, "Delta: Gpu performance model for deep learning applications with in-depth memory system traffic analysis," in *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2019, pp. 293–303.
[7] H. Mo, W. Zhu, W. Hu, G. Wang, Q. Li, A. Li, S. Yin, S. Wei, and L. Liu, "9.2 a 28nm 12.1 tops/w dual-mode cnn processor using effective-weight-based convolution and error-compensation-based prediction," in *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 64. IEEE, 2021, pp. 146–148.
[8] M. Qasaimeh, J. Zambreno, and P. H. Jones, "An efficient hardware architecture for sparse convolution using linear feedback shift registers," in *2021 IEEE 32nd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2021, pp. 250–257.
[9] W. Lou, L. Gong, C. Wang, Z. Du, and Z. Xuehai, "Octcnn: A high throughput fpga accelerator for cnns using octave convolution algorithm," *IEEE Transactions on Computers*, 2021.
[10] Xilinx, *DPUCZDX8G for Zynq UltraScale+ MPSoCs. Product Guide.*, 2022. [Online]. Available: https://docs.xilinx.com/r/en-US/pg338-dpu
[11] D. Wu, Y. Zhang, X. Jia, L. Tian, T. Li, L. Sui, D. Xie, and Y. Shan, "A high-performance cnn processor based on fpga for mobilenets," in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2019, pp. 136–143.
[12] B. Gaide, D. Gaitonde, C. Ravishankar, and T. Bauer, "Xilinx adaptive compute acceleration platform: Versaltm architecture," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2019, pp. 84–93.
[13] Xilinx, *Vitis Unified Software Platform Documentation: Application Acceleration Development (UG1393)*, 2021. [Online]. Available: https://docs.xilinx.com/r/en-US/ug1393-vitis-application-acceleration
[14] ——, *Versal ACAP AI Engine Programming Environment User Guide (UG1076)*, 2021. [Online]. Available: https://docs.xilinx.com/r/en-US/ug1076-ai-engine-environment
[15] G. Alok, "Architecture apocalypse dream architecture for deep learning inference and compute-versal ai core," in *Embedded World Conference*, 2020.
[16] Xilinx. (2021) Vitis ai tool. [Online]. Available: https://github.com/Xilinx/Vitis-AI
[17] P. Chatarasi, S. Neuendorffer, S. Bayliss, K. Vissers, and V. Sarkar, "Vyasa: A high-performance vectorizing compiler for tensor convolutions on the xilinx ai engine," in *2020 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2020, pp. 1–10.
[18] D. Kim, K.-Y. Kim, S. Ko, and S. Ha, "A simple method to reduce off-chip memory accesses on convolutional neural networks," *arXiv preprint arXiv:1901.09614*, 2019.
[19] Xilinx, *Vitis AI Library User Guide (UG1354)*, 2021. [Online]. Available: https://docs.xilinx.com/r/1.4.1-English/ug1354-xilinx-ai-sdk/ZCU102-Evaluation-Kit
[20] M. Tan and Q. V. Le, "Efficientnet: Improving accuracy and efficiency through automl and model scaling," *arXiv preprint arXiv:1905.11946*, 2019.
[21] S. Elfwing, E. Uchibe, and K. Doya, "Sigmoid-weighted linear units for neural network function approximation in reinforcement learning," *Neural Networks*, vol. 107, pp. 3–11, 2018.
[22] Xilinx, *Versal ACAP DSP Engine Architecture Manual (AM004)*, 2021. [Online]. Available: https://docs.xilinx.com/r/en-US/am004-versal-dsp-engine
[23] ——, *UltraScale Architecture DSP Slice User Guide (UG579)*, 2021. [Online]. Available: https://docs.xilinx.com/v/u/en-US/ug579-ultrascale-dsp
[24] ——, *VCK190 Evaluation Board User Guide (UG1366)*, 2021. [Online]. Available: https://docs.xilinx.com/r/en-US/ug1366-vck190-eval-bd
[25] ——, *DPUCVDX8G for Versal ACAPs Product Guide.*, 2022. [Online]. Available: https://docs.xilinx.com/r/en-US/pg389-dpucvdx8g