Ineffectual Neuron Free Deep Learning Accelerator

B. Nikhita Tanvi, Maria Azees, G. Suresh

Abstract*: Convolutional neural network (CNN) is actually a deep neural network which plays an important role in image recognition. The CNN recognizes images similar to visual cortex in our eyes. In this proposed work, an accelerator is used for high efficient convolutional computations. The main aim of using the accelerator is to avoid ineffectusal computations and to improve performance and energy efficiency during image recognition without any loss in accuracy. However, the throughput of the accelerator is improved by adding max-pooling function only. Since the CNN includes multiple inputs and intermediate weights for its convolutional computation, the computational complexity is increased enormously. Hence, to reduce the computational complexity of the CNN, a CNN accelerator is proposed in this paper. The accelerator design is simulated and synthesized in Cadence RTL compiler tool with 90nm technology library.

***Index Terms*: Artificial intelligence, Convolution neural networks, Deep neural networks, Hardware accelerator.**

I. INTRODUCTION

Machine learning offers many pioneering applications such as object detection, smart security, face recognition etc. The image manipulations as well as the computation of image properties are performed by some special hardware processing units called as graphic processing units (GPU). The artificial intelligence (AI) applications like artificial neural network and machine learning processors use AI accelerators for their hardware acceleration [8]. The accelerators used for specialized tasks in the computer systems are known as coprocessors. The extensive applications of GPUs include Robotics, data-intensive tasks, IoTs and so on. However, the large power consumption of GPUs makes the CNNs not suitable for IoTs [11]. Therefore, it is required to design an accelerator for CNNs with high efficiency, less power consumption and high ability and no accuracy loss.

The general purpose processors sometimes cannot meet the performance of the CNN due to its specific computation pattern. To achieve unprecedented accuracy on the application of IoTs the CNNs play a major part[12].For computing general neural network some works are reported for less power consumption to build the good architecture. In some cases the hardware acceleration is targeted by some DNNs. Some applications of artificial intelligence (AI) are hard to run without the network connectivity like cloud based AI applications. The applications such as image recognition, object detection, face recognition are offered by Machine learning [2].The common layer for CNN is pooling function where the function is ignored for accelerating by only concentrating on the convolution part. From the reports it is demonstrated that some CNN accelerators have 168 processing elements in the spatial architecture[7].The results obtained after convolution are fed back to compute the next layer in the accelerator after transferring to CPU/GPU for running the pooling function to be done where the accelerators do not have pooling function[1]-[3]. This consumes much power and also limits the performance. For attaining general neural networks after going through some reports, a neural network with sparsity is proposed to prune the proper network [4].

Basically a CNN is comprised of many layers, there are different types of CNNs like Alexnet, VGGnet, Lenet, googlenet [3] etc. CNNs consists of mainly three layers, one is input layer, output layer, and multiple hidden layers. The convolutional layers of a CNN that convolve with many computations involved with multiplication are the hidden layers. When the filter is convolved with the input data the negative values of the output data are obtained. To eliminate these negative values an activation function ReLu is inserted between the two convolutional layers that is followed by pooling layers and fully connected layers as the inputs and outputs were totally covered by activation function.

In order to detect an object through CNN, the filter weights are required to apply to an image [10]. The process of filtering is the initial stage for image analysis. The image analysis is done by the convolution process of filter weights and the image pixels. Convolutional neural networks work like learnable local filters [5]-[7]. The computation of those pixels is done by linear combination. The edge detection is carried out by providing center pixel with positive weight and the surrounding pixels with negative weights.

The image pixels and the neighborhood of the input layer are connected by the output layer node. The reuse of weights on the different pixels of an image makes the convolutional neural networks unique.

Three main layer types are there to build CNN architectures namely convolutional layer, pooling layer and the ReLu layer.

Convolutional layer: The neural network for each convolutional layer should have following parameters while programming a CNN.

- Input given should be with a shape (number of images) x (width of image) x (height of image).
- The hyper-parameters of the image are height, width of convolutional kernels. The result is passed to the next layer after convolving the input in the Convolutional layers.
- The computation of the output neurons in a convolutional layer is performed by

Revised Manuscript Received on July 09, 2019

B. Nikhita Tanvi, Electronics and Communication Engineering, GMR Institute of Technology, GMR nagar, Rajam, India.

Maria Azees, Electronics and Communication Engineering, GMR Institute of Technology, GMR nagar, Rajam, India.

G. Suresh, Electronics and Communication Engineering, GMR Institute of Technology, GMR nagar, Rajam, India.

- multiplying input pixels present in an image and the weights of the CNN.
- The image features in a convolution are learned by using small boxes like squares of input data, which preserves a good relationship between pixels.

Pooling layer: Sometimes the size of the image is too large to reduce the parameters and to reduce the spatial size. In such scenarios, the pooling layers are introduced periodically between the convolutional layers. The image depth remains unchanged when pooling is performed on the depth dimensions. To reduce the image's spatial size the pooling layer function is performed.

Fully connected (classification) layer: Neurons present in the previous layer is connected to each neuron of the next layer which implies the fully connected layer. Fully connected layers use the function in the output layer called softmax activation function. The features obtained from the output of convolutional and pooling layers are used by fully connected layers to classify the input image based on the training data in various classes.

ReLu(non-linearity): ReLU is a non-linear function which makes the feature map pixel values to zero by replacing all the negative value pixels. ReLu is an element wise operation applied per pixel. To introduce non-linearity in ConvNet, ReLu is activated. The purpose of using ReLU is to introduce a ConvNet which is also non-linear for real-world data.

In this study, we propose a CNN accelerator that can achieve optimal energy efficiency, where the main engine works with many convolutional units for minimizing the ineffectual computations, decomposition of large kernel sized computations and reconfigurable pooling.

The main contribution of this paper includes:

• Designing an accelerator with less degradation or no degradation in the classification accuracy.

• A prototype design with ASIC verification using Cadence with 90nm technology library.

• Computation of the CNN with no accuracy loss.

II. BACKGROUND

A. CNN Basics

Computer vision is the field where many researches have been done in neuroscience since 20 years. Convolutional neural networks are inspired by many researches till today to achieve a good performance as well as computation. In supervised learning algorithms, CNN works as a feed forward process for training and recognition. CNNs are trained in several fields of industrial purposes based on their applications. CNNs achieve more benefiting property in graphical processing units (GPUs) which have high computation ability for unprecedented accuracy on AI applications. When coming to the power limited devices, GPUs and DSPs may face low computation ability, sometimes these GPUs are not suitable for dedicated computation neural network such as CNN. The complex computations consume high power in the CNN accelerator which can be the major drawback for AI applications. We propose an accelerator that works completely for computations to achieve high energy efficiency.

B. Convolutional layer Computation

Basically the computations are performed in the CNN layer. As the name implies, convolution converts the image, convolution layers generate the feature maps from the input images. The working principle of this layer is different than other neural network layers. It doesn't employ connection weights and a weighted sum instead it contains filters that convert images. In connection to this, we call these filters as convolutional filters. The number of feature maps and the number of convolution filters should be same. That means if there are four convolutional filters then it will generate four feature maps.

The filter of convolutional layer is two dimensional matrices.

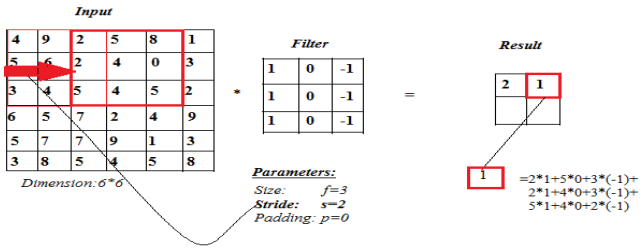

Fig.1 (a) Computation of CNN Convolutional layer

The Computation process of CNN Convolutional layer is shown in Fig.1 (a). In this figure, it is understood that how the convolution works. For instance, let's take a 6*6 pixel image and one convolution filter of size 3*3 arbitrarily. However, actual ConvNet, these filters are determined through training process only. The convolution operation begins at the upper-left of the sub-matrix that is same size as the convolution filter. We applied the convolution to the input and each value in the input is multiplied with the filter and slide all over the input image. Finally, the value obtained after multiplication of the input value with the filter value will be added and replaced by the output block. The 3*3 block is represented by a single value. This process will keep going and the final result will be 2*2 pixel image. The 6*6 pixel image has been converted into a 3*3 pixel image. The convolution yields the large value when the input matches the filter. The feature map extraction depends on the convolution filter what we are using. The layer between convolution and filter is the ReLu activation function in ConvNet. It is same as the function what we used in neural network.

$$O[io][r][c] = B[io] + \sum_{ii=0}^{Fi-1}\sum_{ii=0}^{K-1}\sum_{j=0}^{K-1} \times I[ii][s \times c + j] \times W[io][ii][j] 0 \le io < Fo, 0 \le ii < Fi, 0 \le r < Row, 0 \le c < Col \quad (1)$$

The current output-feature's index number is represented by *io*, the current output-feature's data's row and column number is represented by *r* and *c* respectively as per (1). The total number of output features and input channels are represented by *Fo* and *Fi*. The weight of the filter is represented by channels also called as input features that are present in the input layer. Each filter is represented by the bias weight *B*. In (1), *K*, Row and Col represent the kernel size, output-feature row size and output-feature column size respectively. The output is obtained when computation is done with the inner product of the filter weight and the input data obtained through filter.

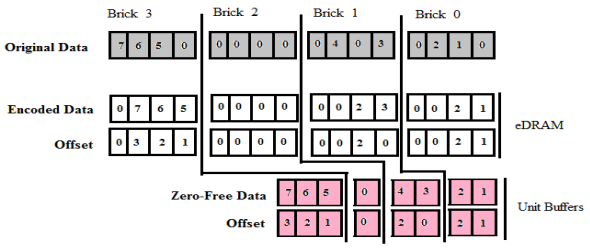

Fig.1 (b) Zero-Free Array Format

Zero-Free Array Format is shown in Fig.1 (b). Zero-free array format is enabled to avoid computations with zero valued data. The zero free array only stores the non-zero values which in turn save the memory and thus performs the effectual computations.

C. Pooling layer

The task of pooling layer is to reduce the size of the image. Pooling layer plays an important role in CNN, it extracts the content from the image pixels in each channel. Pooling layer is divided into two layers as shown in Fig. 2. These layers are namely max pooling layer and average pooling layer. The role of max pooling layer is to select the maximum image data value based on the window called as pooling window. Then, the average value of the data from the pooling window is provided by the average pooling layer.

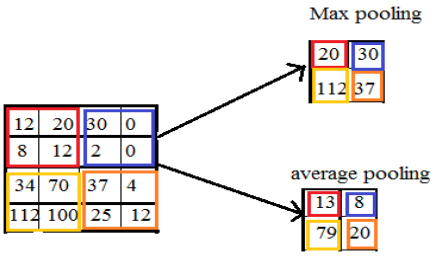

Fig. 2 Pooling layer

Mathematical operation:

O_avg[r][c]= avg [I[r][c] . . . I[r][c+K-1] for average pooling

O_max[r][c] = max[I[r][c] . . . I[r][c+K-1] for max pooling

At position (r, c), I[r][c] represents the input channel data. The kernel size of the window is K.

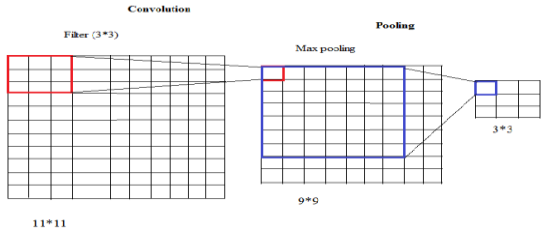

Fig.3 Max Pooling Unit

D. Computational unit

In a CNN computation, the calculation for the inner product of input elements and the filter weights is implemented in the convolutional layer for a three-dimension filter over a three-dimensional input. The product formed by multiplying these filters and weights are then reduced into a single output value using addition. Sometimes, in practice, these values may turn out to be zero; this may take unnecessary calculations for these values. In connection to this, the power and energy performance reduces due to complex computations being performed for the zero valued neurons. For best performance, we need to figure out how many features or values are required. By reducing these zero value features. Our CNN architecture will speed up the inference as well as training. To overcome these challenges we propose a hardware accelerator for CNN to achieve high computations without accuracy loss and optimizing the energy efficiency and high performance. The Computation unit is shown in Fig.4.

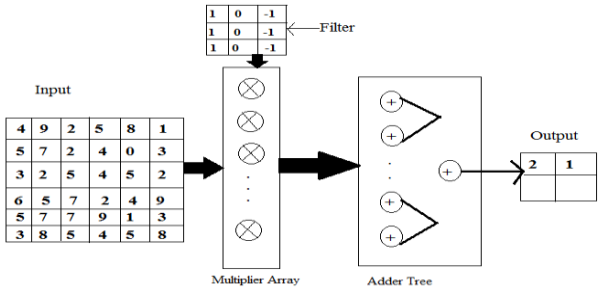

Fig. 4 Computation Unit

III. PROPOSED WORK

A. Reconfigurable Features of CNN accelerator

The proposed CNN accelerator benefits re-configurability and high energy efficiency by using only 3*3 sized computation unit. It supports large kernel-sized filters for computation. When achieving high energy efficiency the hardware cost is thus reduced by minimizing the bus control and module interface through streaming data flow. Obtaining minimum hardware design cost by computing max pooling and average pooling functions of separate pooling blocks with the reuse of convolution engine. To do the computation, the kernel size of the convolution engine must be known, because each convolution engine has its own kernel size ranging from (1*1) to (11*11).When the computation for the convolution with its kernel size is limited above the range, then the accelerator needs to skip its hardware computation to software computation.

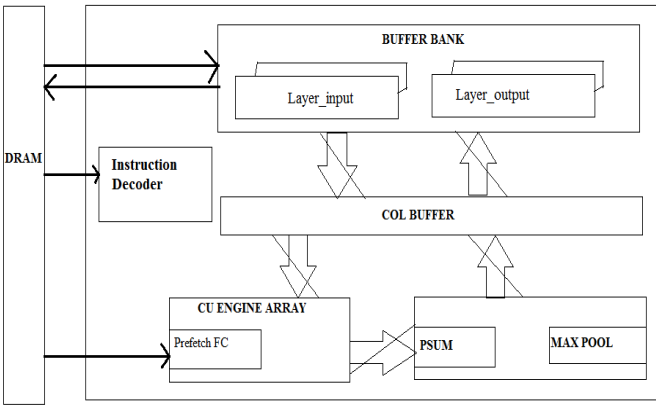

Fig.5 Architecture of CNN accelerator

The architecture diagram of CNN accelerator is shown in Fig.5. To store the intermediate data and data exchange with the DRAM the accelerator has SRAM as a buffer bank with 96kbytes.To store the input data of the present layer and output data, buffer bank is divided into two sets as Bank A and Bank B. The odd-number channels are stored in Bank A and the even-number channels are stored in Bank B. To remap the buffer bank's output to the input of the convolution engine a Col Buffer module is implemented. To enable high parallel computation, 16 convolutional units are composed in CU engine. From the Direct Memory Access controller, a pre-fetch controller is embedded in the engine to fetch the parameters periodically and to update the weight and bias values in the engine. Coming to the Scratchpad, it accumulates and stores the partial convolution results by working together with accumulator. The ACCU buffer is embedded with separate max pooling module to pool the data of the output-layer. The (AXI) bus is an Extensible Interface which is 16-bit advanced to control the accelerator. When enabling the accelerator, the DRAM stores the processed CNN commands in advance and is loaded automatically to a 128-depth command FIFO.

The convolution starts when the image scratchpad present in the ACCU Buffer is reset. The upcoming layer's property like numbers and channel size are configured by two types of commands namely configuration commands and execution commands. These commands are inserted in between the multiple layers which enable the max pooling and ReLu functions. Initiation of the convolution or pooling computation includes the execution commands. The data of the input-layer is sent to convolution unit (CU) engine. The output feature filters the weight and the inner products of channel's data which are calculated by the CU engine. The output features which are sent to the Buffer bank is considered as the accumulated image in the scratchpad. The CU engine output results are passed to the ACCU Buffer block with the accumulated stored results in the scratchpad. For generating the next output feature, the convolution process will be duplicated by the CNN accelerator from DRAM. DRAM already has the updated filter weights after completing the computation of the 1st feature. This process is repeated for calculation of all the remaining features.

IV. METHODOLOGY

When we consider the neural networks like CNN/DNNs, the architecture is built based on the performance as well as the application which is implied. In most of the CNNs, the challenges are aroused based on the operations which are involved in the computation process. To overcome these challenges, in this paper, an accelerator is implemented with low cost, high performance and high energy efficiency. Design of the hardware accelerator for CNN inference includes implementation of Convolution, pooling and activation layers on specialized hardware. The CNN functionality is verified with the help of the state-of-the-art accelerators like LeNet or Alexnet or Vgg-net with some datasets based on the requirement. The functionality of CNN is performed prior to the hardware implementation, in the training stage and then a model is developed by processing the dataset of any one of the accelerator through the CNN. The CNN architecture is built as stack of layers. Each layer of CNN is implemented as an individual function. The output of each layer is a set of feature maps containing the features extracted in that layer. The output of the final layer is used to calculate the error by comparing with the labels of the dataset. At the end of the training, a model with updated weight vectors is attained.

We designed the accelerator by implementing each block with Verilog module with the help of Cadence suit and then simulation and synthesis are performed. The design is synthesized using Cadence RTL compiler with a 90nm technology library. The simulation is performed for the entire hardware and then each block is simulated individually and the waveforms are generated with respect to the input and outputs given. Timing and power reports are obtained after proper synthesis in the RTL compiler tool and then delay is calculated.

For the design of hardware accelerator, the two main modules such as CU engine and Max pooling are discussed in the following sections.

A. CU Engine

While composing the CU engine, nine multipliers are used by the accelerator. The Processing engine (PE) is included in CU engine array. Through a D flip-flop is used to pass the input data to the next stage of PE. Based on the input data, the PE performs multiplication function for the filter's weight and input data. To combine with the output, the CU engine array adds nine Processing engines (PE) outputs using an adder. When the convolution stride is more than one, the Computation power can be saved by turning the multiplication function on and off. The result obtained after multiplication during the 3*3 convolution is sent to the adder to deliver the added result to the final output in the CU. The Filter weights are stored from the global bus in the CU through the DMA controller which is fetched from DRAM. The CU updates the filter weights at the Processing Engines input only after the request signal is sent to the input channel.

B. Max Pooling

From the output feature, the eight rows of data is stored in the scratchpad in the parallel manner. To access simultaneously, the data share one column address at a time. The data that is stored in the scratchpad will not be valid if there is any difference in the stride size in the convolution. If the stride is equal to 2, only R0, R2, R4, R6 are stored and then the data is validate. The kernel size of the pool window may also be configured to 2 or 3. The validated input data is selected when MUX is set in front of the max pooling module for accommodating

pool-size and different strides of convolution. The Fig.6 shown below is the Max pooling module architecture.

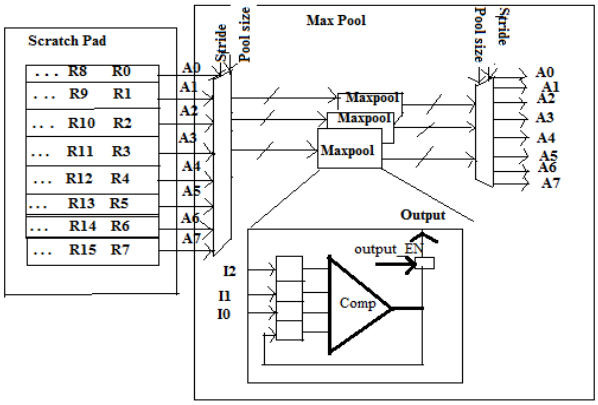

Fig.6 Architecture of Max pooling module

V. PERFORMANCE ANALYSIS

We designed a hardware accelerator and it is simulated using Cadence NC launch tool with Verilog code. The design is elaborated and synthesized in Cadence RTL compiler tool which generates the verilog code to the gate level netlist. The physical design is to place and route each block of the design and it is implemented and verified with the help of Cadence SoC Encounter Tool. The technology library used in this paper is gpdk090bc (90nm). RTL design synthesis gives report on power and timing analysis.

RTL Simulation and Schematic Results

Prior to the design of the hardware accelerator, the functionality verification for each block in the architecture of the CNN accelerator is checked. For high-level representation of the circuit, Register-Transfer-Level abstraction (RTL) is implemented for the accelerator design and the inner blocks of the accelerator. To observe the power and timing report, synthesis of the accelerator is processed using Cadence RTL compiler tool. To simulate the verilog code, we use NC launch tool with 90nm technology library. The simulated and synthesized designs in addition to the physical verification of the CNN accelerator along with its modules are shown in the following.

RTL Schematic of Convolutional Unit: The convolution layer is implemented in the CU engine. Fig.7 shows the RTL schematic of single convolution unit synthesized in Cadence RTL compiler.

Fig.8 shows the RTL simulation result of a CU engine. All samples in the input data, filter weights and output data are in the 16-bit fixed point binary numbers.

RTL Design of Pooling block: The Pooling block is divided into four Pooling units. Each unit process has two rows of data. The Schematic diagram of a Pooling unit is shown in Fig.9.

Simulation of pooling unit: Two input samples are compared using a 16-bit comparator and the output is the intermediate value which is stored in the Parallel-in-Parallel-out (PIPO) register. This value is stored in the third input in the next clock cycle. The output of the pooling unit is read at every clock cycle. Fig.10 shows the simulation result obtained after compilation of a single pooling unit.

RTL Schematic of CU engine: There are two convolution engines in CU engine array, each convolution engine comprises of eight convolutional units. The overlapped data of rows can be processed simultaneously since it is a combinational circuit. Through the COL buffer and pre-fetch blocks, the filter weights and input data that are present in memory bank, are passed to the CU engine. Fig.11 shows the RTL schematic of CU engine obtained after compilation with 90nm technology library.

RTL Schematic of ACCU Buffer: The output results from the CU engine are accumulated using ACCU Buffer where the CU engine stores the output feature data in the scratchpad. Then, the max pooling computation is done in separate pooling block in ACCU Buffer. Fig.12 shows the compilation result obtained for ACCU Buffer.

RTL Schematic of Buffer Bank: The synthesized design of Buffer bank is implemented in RTL compiler in Cadence which stores the even and odd number of channels divided as Bank A and Bank B in Buffer bank shown in Fig.13.

Block	Leakage Power(mW)	Dynamic Power(mW)	Total Power(mW)
Buffer bank	3.007799	58.534512	58.543022
CU engine array	0.140015	12.021756	12.143451
COL Buffer	0.001560	9.743652	9.546721
Accumulator	0.023022	1.110211	1.110621
Max Pool Block	0.00042	0.555632	0.556474
Total Hardware	3.172816	81.965763	81.900289

Table 1: Power report of the accelerator

From the above table, we figure out that the maximum power consumption of memory banks. This occurs due to the several read/write operations in the convolution and pooling stages. Since the convolution engine performs convolution operation, the CU engine array consumes more power.

Delay of the CNN accelerator: After the synthesis of the design, Cadence RTL compiler yields the timing report of the design. The total delay in the accelerator is observed as 8.01 ns.

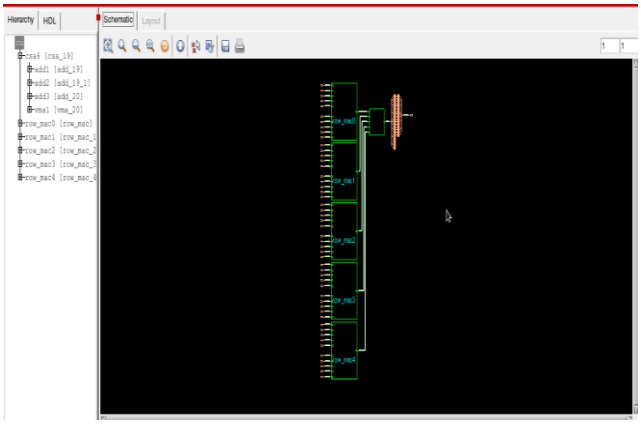

Fig .7 RTL Schematic of Convolutional Unit

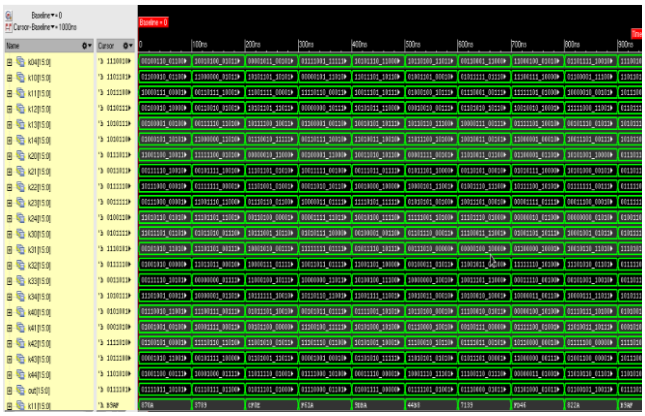

Fig.8 Simulation results of Convolutional unit

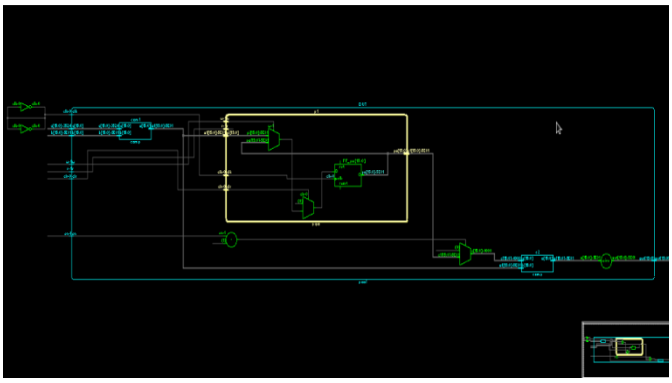

Fig.9 Schematic of Pooling Unit

Fig.10 Simulation results of Pooling Unit

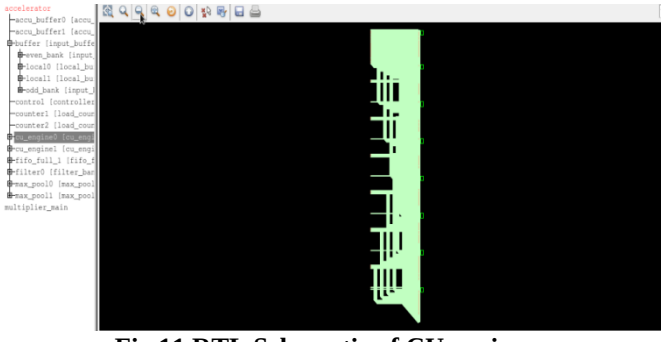

Fig.11 RTL Schematic of CU engine

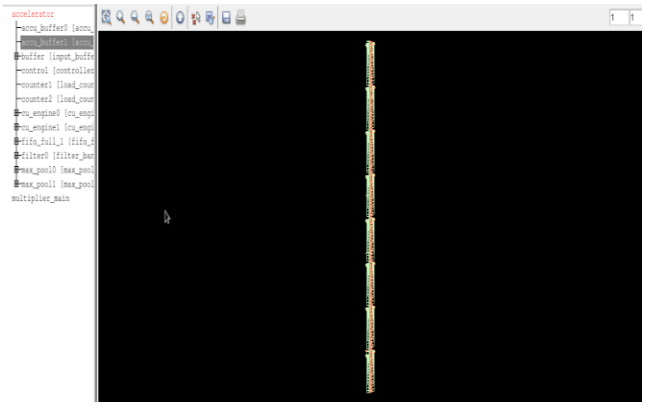

Fig.12 RTL Schematic of ACCU Buffer

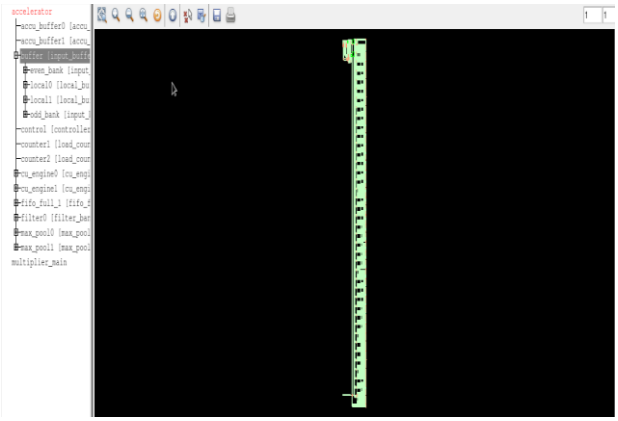

Fig.13 RTL Schematic of Buffer bank

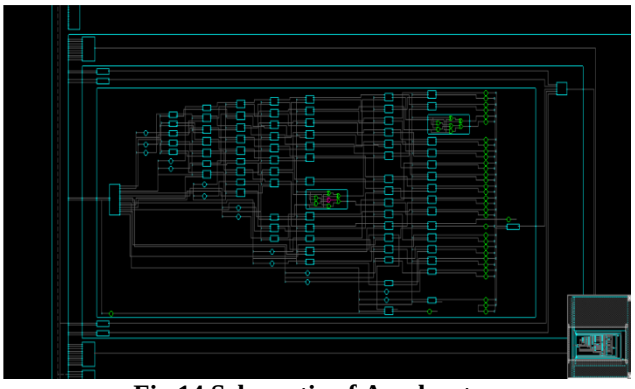

Fig.14 Schematic of Accelerator

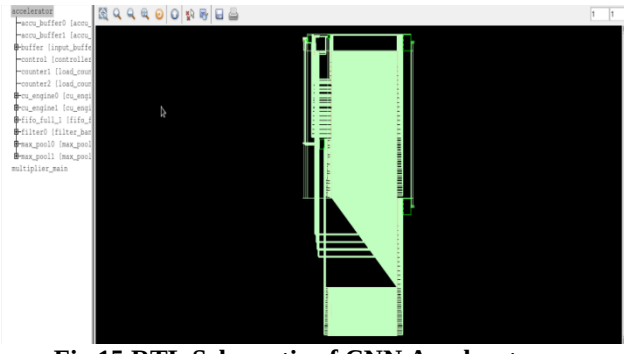

Fig.15 RTL Schematic of CNN Accelerator

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

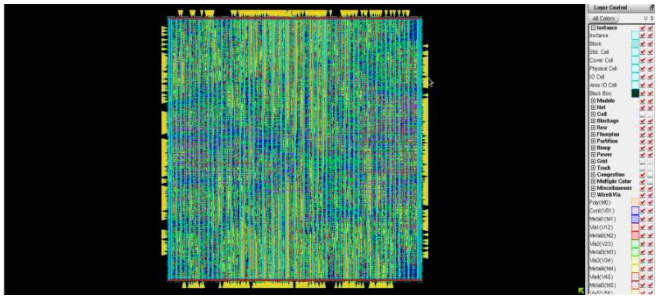

Fig.16 Physical Design of Convolutional Unit

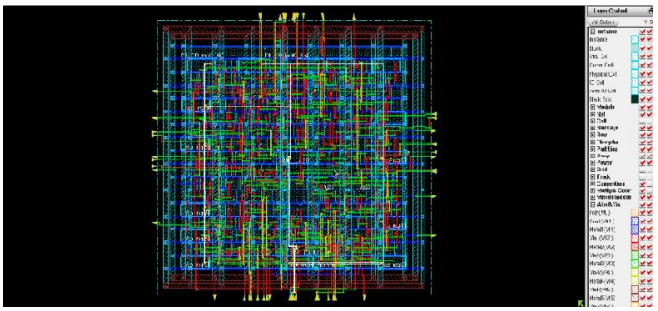

Fig.17 Physical Design of Pooling Unit

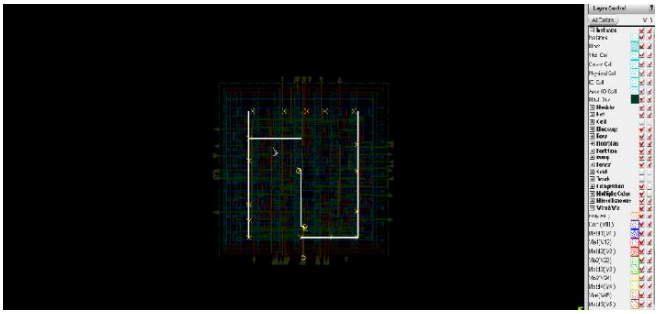

Fig.18 Clock obtained for Pooling unit

VI. CONCLUSION

In this paper we propose a CNN accelerator which works with high energy efficiency, low cost and high performance. Sometimes there will be complex computations in the hardware that may lead to the accuracy loss in the accelerator and causes the increment in the output bandwidth for CU engine. To avoid this type of ineffectual computations we proposed a hardware CNN accelerator. This accelerator supports the performance of both pooling block and CU engine. The CU engine implements the convolutional layer. The accelerator takes the image data and trained weight vectors as inputs. The accelerator is reconfigurable as it takes any number of input images for different training models. The functionality of the design is verified by training the CNN with a dataset. The hardware of the accelerator is designed with a Verilog code in NC launch tool and the code is written alone for each block in the architecture to prove the performance ability of the CNN accelerator. A 90nm technology library is used for synthesizing the design to obtain power and timing reports, where the total power consumption is 81.90mW with the delay of 8.01ns. The simulation of the design is performed with the help of NC launch tool. The data is a 16-bit fixed point format throughout the design. The major factor affecting the speed of the accelerator is the memory access speed. We implemented buffer banks which helps in storing the data of the input and filter weights. The physical verification for convolutional unit and pooling unit is verified using Cadence SoC Encounter tool.

REFERENCES

1. S. A. Moshovos, J. Alberico, P. Judd, "Value-Based Deep-Learning Acceleration",vol.38, pp.41-45, Feb. 2018 in IEEE Micro.
2. Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521,pp. 436–444, May 2015.
3. R. Hameed *et al.*, "Understanding sources of inefficiency in general-purpose chips," in *Proc. 37th Annu. Int. Symp. Comput. Archit.*, 2010, pp. 37–47.
4. S. Han *et al.*, "EIE: Efficient inference engine on compressed deep neural network," in *Proc. 43rd Int. Symp. Comput. Archit.*, 2016, pp. 243–254.
5. M. Horowitz, "Computing's energy problem (and what we can do about it)," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2014, pp. 10–14.
6. J. Albericio et al., "Cnvlutin: Ineffectual-Neuron-Free Deep Neural Network Computing", *43rd ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2016, [online] Available: http://ieeexplore.ieee.org/document/7551378/.
7. Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," in *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers (ISSCC)*, San Francisco, CA, USA, Jan./Feb. 2016, pp. 262–263.
8. A. Shafiee *et al.*, "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *Proc. 43rd Int. Symp. Comput. Archit.*, 2016, pp. 14–26.
9. P. Judd et al., *Reduced-Precision Strategies for Bounded Memory in Deep Neural Nets*, 2015, [online] Available: https://arxiv.org/abs/1511.05236.
10. A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet classification with deep convolution neural networks," in *Proc. Conf. Workshop Neural Inf. Process. Syst.*, 2012, pp. 1106–1114.
11. S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proc. 32nd Int. Conf. Int. Conf. Mach. Learn. (ICML)*, vol. 37. Lille, France, 2015, pp. 1737–1746.
12. K. Simonyan and A. Zisserman, "Very deep convolution networks for large-scale image recognition," *CoRR*, pp. 1–14, Sep. 2014.

AUTHORS PROFILE

B.Nikhita Tanvi received her B.Tech degree in Electronics and Communication Engineering from Sri Sivani College of Engineering which is affiliated under JNTUK, Andhra Pradesh, India in 2016 .She is pursuing her M.Tech degree in VLSI and Embedded Systems Design specialization in GMRIT which is affiliated under JNTUK, Andhra Pradesh, India. Her areas of interest are ASIC design implementations and low power VLSI.

Maria Azees received the B.E. degree in electronics and communication engineering and the M.E. degree in applied electronics from St. Xavier's Catholic College of Engineering, Nagercoil, India, which is affiliated under Anna University, Chennai, India, in 2011 and 2013, respectively. He received his Ph.D. degree in the faculty of information and communication engineering from Anna University, Chennai, in 2017. He is currently working as the senior Assistant professor in GMR Institute of Technology, Andhra Pradesh, India. His research interests include security in wireless sensor networks and VANETs.

Gulivindala Suresh is working as an Assistant Professor in the Department of ECE, GMR Institute of Technology, Rajam, Andhra Pradesh, India. He obtained his M.Tech from Biju Patnaik University of Technology, Odisha, India. He obtained his B.Tech from JNTU, Andhra pradesh, India. He is presently pursuing his Ph.D. in the Department of ECE at JNTUK University, Andhra Pradesh, India. His research interests include Digital Image Processing and VLSI technologies. He is a member of IETE.