Bit-Serial Computing Technique based Efficient Deep Neural Network Accelerator

MS (Research) Thesis

By

HARSH CHHAJED

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

JUNE 2021

Bit-Serial Computing Technique based Efficient Deep Neural Network Accelerator

A THESIS

Submitted in fulfillment of the
requirements for the award of the degree
of

Master of Science (Research)

by

HARSH CHHAJED

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY INDORE

JUNE 2021

INDIAN INSTITUTE OF TECHNOLOGY INDORE

CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the thesis entitled **Bit-Serial Computing Technique based Efficient Deep Neural Network Accelerator** in the fulfillment of the requirements for the award of the degree of **MASTER OF SCIENCE (RESEARCH)** and submitted in the **DEPARTMENT of ELECTRICAL ENGINEERING, Indian Institute of Technology Indore**, is an authentic record of my own work carried out during the time period from July 2019 to June 2021 under the supervision of Dr. Santosh Kumar Vishvakarma, Associate Professor, Indian Institute of Technology Indore

The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other institute.

15/06/2021

Signature of the student with date Harsh Chhajed

This is to certify that the above statement made by the candidate is correct to the best of my/our knowledge.

15.06.2021

Signature of the Supervisor of
MS (Research) thesis (with date)
Dr. Santosh Kumar Vishvakarma

- **Harsh Chhajed** has successfully given his MS (Research) Oral Examination held on 27/08/2021

27/08/2021

Signature of Chairperson (OEB) with date

27/08/2021

Signature(s) of Thesis Supervisor(s) with date

27/08/2021

Signature of Convener, DPGC with date

27/08/2021

(HOD-EE, Officiating)
Signature of Head of Department with date ---

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my gratitude to people who, in one or the other way, contributed by making this time learnable, enjoyable, and bearable. At first, I would like to thank my supervisor **Dr. Santosh Kumar Vishvakarma**, who was a constant source of inspiration during my work. With his constant guidance and research directions, this research work has been completed. His continuous support and encouragement has motivated me to remain streamlined in my research work.

I am thankful to **Dr. Srivathsan Vasudevan** and **Dr. Mrigendra Dubey**, my research progress committee members, for taking out their valuable time to evaluate my progress all these years. Their valuable comments and suggestions helped me to improve my work at various stages.

My sincere acknowledgment to Indian Institute of Technology Indore for providing me the opportunity to explore my research capabilities.

I would also like to thank all the members of Nanoscale Devices, VLSI Circuit System Design Lab (NSDCS) research group, especially to **Mr. Gopal Raut**, **Mrs. Neha Gupta** and **Mr. Narendra Singh Dhakad** for discussions and help during my thesis work.

I would like to express my heartfelt respect to my parents for their love, care and support they have provided to me throughout my life.

Harsh Chhajed

Dedicated

to

My Parents

ABSTRACT

Contemporary hardware implementations of deep neural networks face the burden of excess area requirement due to resource-intensive elements such as a multiplier. A semi-custom ASIC approach-based VLSI circuit design of the multiply-accumulate unit in a deep neural network faces the chip area limitation. Therefore an area and power-efficient architecture for the multiply-accumulate unit is imperative to down the burden of excess area requirement for digital design exploration. The present work addresses this challenge by proposing an efficient processing and bit-serial computation based multiply-accumulate unit implementation. The proposed architecture is verified using simulation output and synthesized using Synopsys design vision at *180nm* and *45nm* technology and extracted all physical parameters using Cadence Virtuoso. At *45nm*, design shows 34.35% less area-delay-product (ADP). It shows improvement by 25.94% in area, 35.65% in power dissipation, and 14.30% in latency with respect to the state-of-the-art multiply-accumulate unit design. Furthermore, at lower technology node gets higher leakage power dissipation. In order to save leakage power, we exploit the power-gated design for the proposed architecture. The used coarse-grain power-gating technique saves 52.79% leakage/static power with minimal area overhead. The system-level performance of MAC has evaluated using LeNet architecture which is implemented on the FPGA board to validate the performance parameters impact of the proposed multiply-accumulate unit. The architecture with the proposed multiply-accumulate unit saved 49.39% look-up table utilization for multiply-accumulate unit and 20% overall look-up table utilization compared to MAC with Xilinx multiplier and adder IP. The proposed architecture over performed for critical time delay compared to the state-of-the-art.

Contents

List of Figures

List of Tables

List of Abbrevations

DNN	:	Deep neural network
MAC	:	Multiply-accumulate
AF	:	Activation function
CPU	:	Central processing unit
GPU	:	Graphics processing unit
FPGA	:	Field programmable gate arrays
ASIC	:	Application specific integrated circuit
DSP	:	digital signal processing
IOT	:	Internet of things
CNN	:	Convolutional neural network
PG	:	Power gating
MOS	:	Metal-oxide-semiconductor field-effect transistor
TPU	:	Tensor processing unit
LSB	:	Least significant bit
MSB	:	Most significant bit
EDA	:	Electronic design automation
IP	:	Intellectual property
ReLU	:	Rectified linear unit
SoC	:	System on chip
HDL	:	Hardware description language
RTL	:	Register transfer level
RAM	:	Random access memory
ROM	:	Read only memory

VLSI	:	Very large scale integration
CMOS	:	Complimentry metal oxide semiconductor
TT	:	Typical typical
IDE	:	Integrated development environment
BRAM	:	Block random access memory
LUT	:	Look up table
ADP	:	Area delay product
AI	:	Artificial intelligent

Chapter 1

Introduction

1.1 Overview

A deep neural network (DNN) has significantly improved computing paradigms within the last decade. The main advantage of a DNN is that it can learn the hidden relationships in data with irregularity [2]. Efficient DNN architectures has been proposed for different applications like image classification, speech recognition [3], computer vision [4] and natural language processing [5]. The neural network benefits every field, whether it is a defense sector, health sector, or automation sector. However, with the increasing demand for DNN, the computational intensity and complexity increased significantly, resulting in high power and more logical resource utilization [6,7]. Further, in the DNN, each neuron performs two primary functions: accumulating the weighted features using multiply-accumulate operation (MAC) and evaluate the activation function (AF) from the accumulated sum.

There are different hardware platforms like central processing unit (CPU), graphics processing unit (GPU), application-specific integrated circuit (ASIC) or field-programmable gate arrays (FPGA) on which DNN can be implemented. The disadvantage of a general-purpose platform such as a CPU is that it fails to exploit the parallelism of a neural network. Also, the resource utilization is low, which results in low performance and high power dissipation [8]. GPU uses parallelism in the network, but it also has very high power consumption. On the other end of the spectrum, the

hardware-based neural network has the advantage of faster execution time than the software-based neural network. Moreover, ASICs have specialized hardware structures for MAC and AF computation, and thus they achieve high resource utilization and lower power consumption than other mentioned platforms [9–11].

An ASIC has a fixed computation design architecture. Moreover, a configurable design architecture is preferred that scales well with deep neural network size. Further, resources efficient approximate logic design for neural network computation also investigated where can compromise with performance accuracy [12]. In comparison, some techniques have a nice balance between area and functionality without compromising accuracy. Different variants of deep neural network implementation are based on network topology, logic implementation, and neuron computational architecture. In all of these variants, the common and the most repetitive unit in the network is the MAC unit, also known as the processing element. The MAC unit is responsible for more than 90% of the overall computational power of the neural network [13]. Therefore, the MAC computational unit architecture needs to be optimized to increase the physical performance parameter of the deep neural network in terms of the area, power and delay of the overall circuit.

The rest of this chapter is organized as follows. The motivation behind our work is elaborated in Section 1.2. In Section 1.3, we present the summary of thesis contributions and outline of the rest of thesis is described in Section 1.4.

1.2 Motivation

When it comes to full-scale hardware implementation of the DNN on FPGA/ASIC, the designer has to make some choices like bit precision after studying all the trade-offs in the architecture. Further, if the design is implemented in FPGA, digital signal processing (DSP) blocks are not preferred due to very high power utilization [14] as the DSP in Xilinx 700 series FPGAs are 48-bit precision computation unit and using a 48-bit unit for 12 or 16-bit operation is very inefficient. Further MAC unit is the sole computational element in the DNN and consumes most of the hardware resources.

After several trade-offs studied in the MAC computation, the designer can use the bit-serial computation technique in the processing element. Artificial neural networks (ANN) and CNN are resource-hungry, and hence it requires resource-efficient and power-efficient design architecture for low power IoT applications.

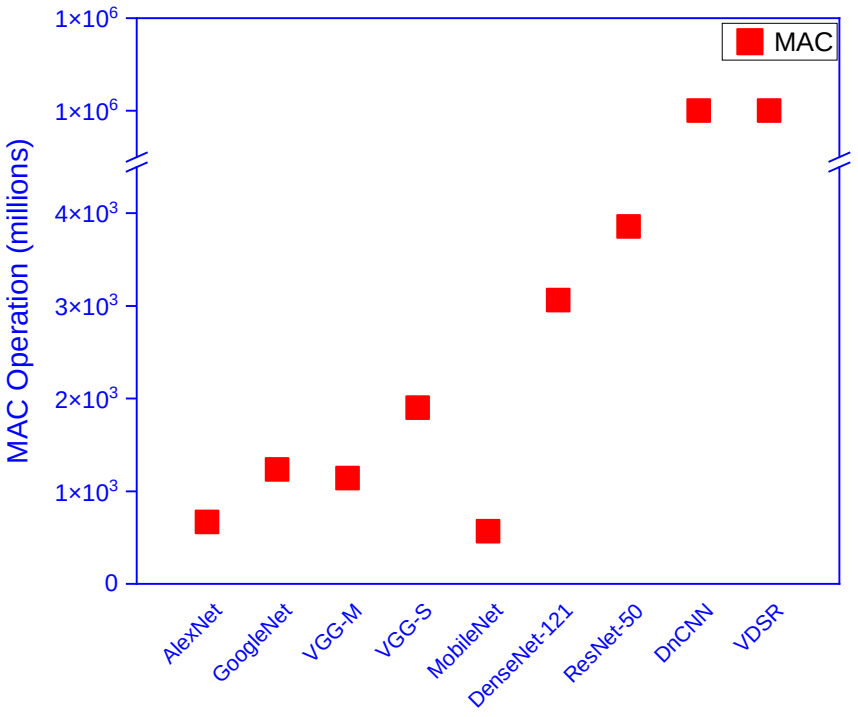


Figure 1.1: Number of MAC operation required in different CNN architecture [1]

.

In the last decade, the number of layers of DNN for different applications has increased significantly as deeper networks are better at learning more complex relations. Different techniques and architectures are proposed in the state-of-the-art [15, 16] for making a more accurate network.The number of MAC operations required in the different well-known CNN architectures are shown in the Figure 1.1. The CNN architectures are used for different computer vision and computational imaging applications like image classification, image de-noising and super resolution. The Table 1.1 shows the numerical figures, dataset and application for different CNN networks. However, they need more hardware architecture and not feasible for low power and area applications.

Table 1.1: CNN Benchmarks

Network	Number of convolutional layer	Number of MAC operation (in millions)	Dataset	Application
AlexNet	5	665.8	ImageNet (224*224)	Image Classification
GoogleNet	57	1233	ImageNet (224*224)	Image Classification
VGG-M	5	1141	ImageNet (224*224)	Image Classification
VGG-S	5	1901.5	ImageNet (224*224)	Image Classification
MobileNet	27	567.7	ImageNet (224*224)	Image Classification
DenseNet-121	120	3062	ImageNet (224*224)	Image Classification
ResNet-50	53	3855.9	ImageNet (224*224)	Image Classification
DnCNN	20	1380000	FHD Images (1920*1080)	Image De-noising
VDSR	20	1380000	FHD Images (1920*1080)	Super-resolution

Therefore, it motivates us to work on the efficient MAC design, which can be power and area efficient and helpful when the area and the power are on a tight budget. To address this issue, we have implemented an elegant bit-serial computing-based MAC unit for DNN applications. The proposed technique is feasible for both ASIC and FPGA applications. The proposed design is both area and power-efficient. However, to make more power efficient, we use the power gating technique in the MAC architecture, which reduces static power, which is also the key concern at lower technology node.

1.3 Thesis Contribution

Traditionally MAC architecture is implemented using the Bit-parallel computing technique, which increases the hardware requirements quadratically as the bit precision increases [17]. We optimize the overall circuit design using the proposed bit-serial computing-based MAC architecture to address the trade-off between area, power and delay. In this thesis, the efficient logic architecture with lowering the static power dissipation scheme is proposed. We have designed an N-bit precision MAC unit using a bit-level processing technique to improve the overall physical parameters like area, power and delay of the circuit, and the power gating scheme is used for further static power saving. Subsequently, using the proposed MAC, we have implemented the

LeNet CNN architecture on the FPGA board. The key points of the proposed work are as follows:

- A semi-custom digital design approach for the MAC unit is investigated for DNN applications.
- A bitcell architecture for a 1-bit multiplication calculation is proposed. Further, the power-gating (PG) approach is employed and evaluate with *180nm* and *45nm* technology node to demonstrate power savings.
- An 8-bit MAC unit is designed using the proposed power-gated bitcell and the impact on the MAC performance is analyzed for parameters like area, power and delay.
- The performance of the proposed MAC is validated in terms of physical parameters for different bit precision like 8-bit, 12-bit and 16-bit. The performance analysis is evaluated at higher(*180nm*) as well as lower(*45nm*) technology node.
- The LeNet architecture is designed using the proposed bitcell-based MAC, and the network's performance is analyzed.

1.4 Organization of Thesis

The rest of this thesis is organized as follows:

Chapter 2: In this chapter, we discuss the DNN background and the related architectures. Furthermore, the critical components of the neural network architecture and different techniques to implement the critical units are also explained.

Chapter 3: In this chapter, we describe the bitcell-based architecture for the implementation of the MAC unit. Further, the bitcell architecture using power gating is discussed. Then the N-bit precision MAC unit has been discussed using the bitcell architecture.

Chapter 4: In this chapter, we describe the complete LeNet architecture. First, each layer of the network is explained. Then the complete hardware implementation of the

architecture is explained.

Chapter 5: In this chapter, we describe both design approaches. First the semi-custom ASIC approach for the multiply-accumulate unit. Second the FPGA-based approach for the deep neural network (LeNet). Finally, both designs approach design flow and tools are explained in detail.

Chapter 6: In this chapter, we summarize the results and discuss the various parameters of the proposed architecture, and compared it with the state-of-the-art.

Chapter 7: In this chapter, we give the conclusion of the work and provide the direction for future work.

Chapter 2

Background and Related Work

This chapter introduces a DNN, its basic architecture, and its applications. Then we discuss the basic MAC unit function and different computing styles. Subsequently, the architecture of different state-of-the-art multiplier is discussed.

2.1 Deep Neural Network

Deep learning uses the DNN to perform various machine learning tasks. The neural networks are the combination of neurons connected with each other. There are different layers of neurons in the network. Neurons in one layer are not connected. Each neuron generates an output based on the input from the previous layer neuron. The output is the weighted sum of the inputs followed by a nonlinear activation function (AF). The first layer neurons are called input neurons, and the last layer neurons are called output neurons. If the layers are more between the input and output neurons, then the network is called a DNN. As such, there is no strict definition of the number of layers, but in general, if it is more than seven, the network is called DNN [18].

DNN advantage over other machine learning algorithms is the hierarchical nonlinear processing over the multiple layers. In the initial layers of the network, simple features of the data are extracted. Then, as we go deep in the network, complex features are extracted using the simple features that create the hierarchical representation

of the data.

There are different types of DNN which is built for different application purpose. However, some components are constant in all networks, like neurons, weight, bias, and MAC computation. The two widely used networks are explained below.

1. Fully Connected Artificial Neural Network (FCANN) - It consists of multiple layers in which the first layer is the input layer, the last layer is the output layer, and all the middle layers are called hidden layers as shown in Figure 2.1. This network is the analogy of the biological neural networks present in the brain. Each neuron layer, after calculating the weighted input sum, is applied to a nonlinear AF. ANN is used in various applications like pattern recognition, clustering, classification, and many prediction areas [19]. It is responsible for more than 61% Google TPU's workload [20].

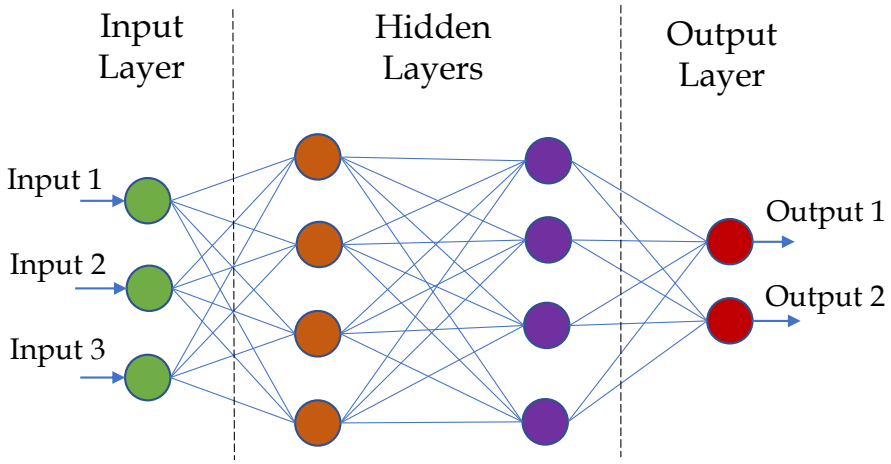


Figure 2.1: Fully Connected Artificial Neural Network

2. Convolutional Neural Network (CNN) - CNN is the most widely used neural network nowadays, especially in vision-related applications. It is very effective in applications where input data has a grid-like topology [21]. A typical CNN network is shown in Figure 2.2. It mainly consists of three layers, convolution layer, pooling layer, and fully connected layers. The convolution layer extracts the features from the data using the convolution operation. The pooling layer

downsamples the output features from the convolution layer and reduces the size. Finally, the fully connected network performs the classification task.

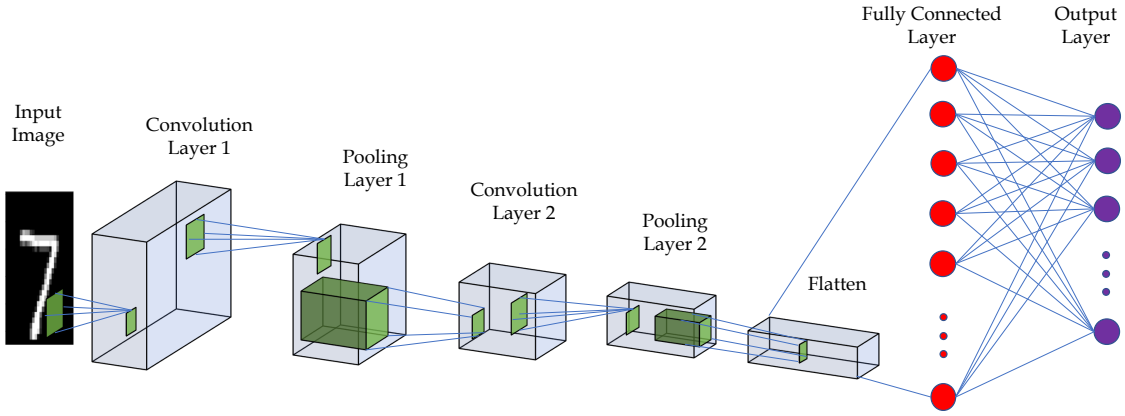


Figure 2.2: Conventional Convolutional Neural Network

The central computation unit in both the network is the MAC unit. It is explained in detail in the following section.

2.2 Multiply-accumulate unit

The fundamental artificial neuron mimicking the biological neuron is shown in Figure 2.3. As shown, the dendrite works as an input to the neuron. A single neuron in the fully connected network has input equal to the total number of outputs/ neurons in the previous layer. The synapse consists of the weights, and also it multiplies the input and weight. The neuron finally accumulates all the weighted inputs, adds the bias, and applies the AF to give the neuron's output.

In ANN the synapse multiplication and neuron accumulation operation is known as the MAC function. The basic MAC unit consists of multiplier, adder, and accumulator blocks. The MAC computational unit shown in Figure 2.4. The input and output arithmetic relation in the preceding layer of a fully connected neural network is shown below:

$$a_j^l = f(\sum \omega_{jN}^l a_N^{l-1} + b_j^l) \tag{2.1}$$

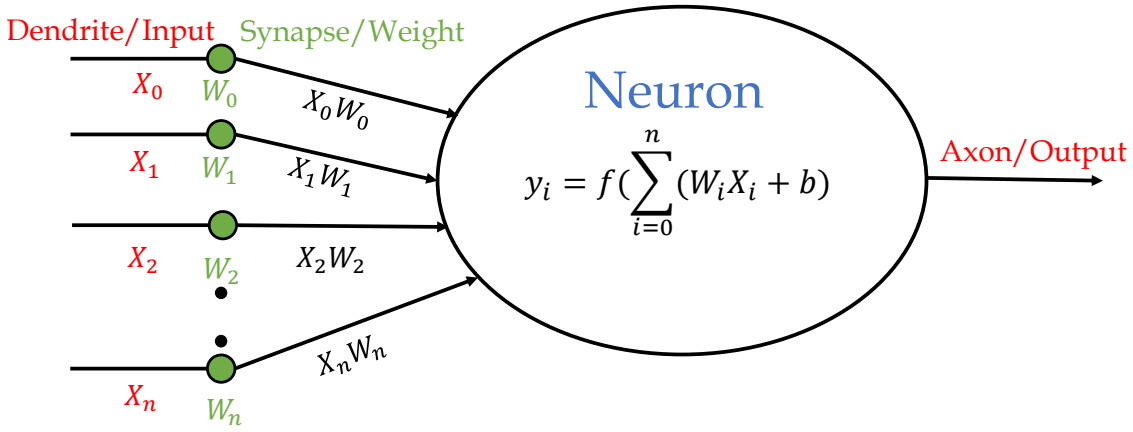


Figure 2.3: Block level single artificial neuron architecture mimicking the biological neuron

where function f is the activation function (tanh/sigmoid/ReLu) of the computational unit k, corresponding to overall neurons in the $(l-1)^{th}$ layer. To formulate this equation in matrix form, we assume a weight matrix ω^l corresponding to layer l. Elements of the weight matrix ω^l are weighted to the inputs of neurons from l^{th} layer with j^{th} row and N^{th} column. b^l_j is the bias here. Each MAC unit consists of a j^{th} multipliers followed by an adder tree.

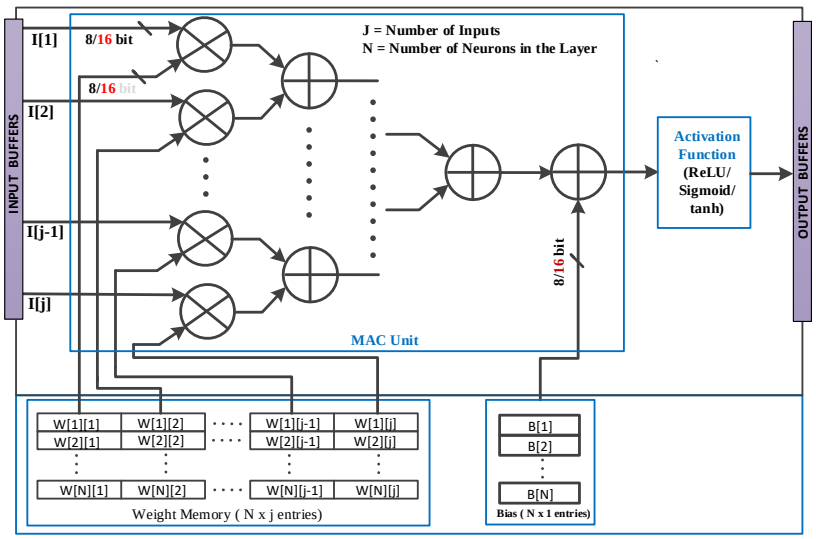


Figure 2.4: Multiply-accumulate unit with parallel multiplier and activation function for N neurons in a layer

The dimensions of the matrix in equation 2.1 are as follows :

w^l - is the weight matrix in l^{th} layer of dimension $j \times N$ where j is the number of inputs from $(l-1)^{th}$ layer and N is the number of neurons in the l^{th} layer

a^{l-1} - is the input matrix of dimension $1 \times j$, where j is the number of inputs from $(l-1)^{th}$ layer

b^l - is the bias matrix in l^{th} layer of dimension $1 \times N$ where N is the number of neurons in the l^{th} layer

a^l - is the output matrix from l^{th} layer of dimension $1 \times N$, where N is the number of neurons in the l^{th} layer. It will work as a input matrix to the $(l+1)^{th}$ layer.

For example there are j input coming from 1^{st} layer and the 2^{nd} layer has 3 neurons. As per the equation 2.1 the value for j=j, N=3, and l=2. The matrix computation in this case will be given as

$$a^l = \begin{bmatrix} I[1] & I[2] & \dots & I[j] \end{bmatrix} \begin{bmatrix} w_1[1] & w_2[1] & w_3[1] \\ w_1[2] & w_2[2] & w_3[2] \\ \vdots & \vdots & \vdots \\ w_1[j] & w_2[j] & w_3[j] \end{bmatrix} + \begin{bmatrix} B[1] & B[2] & B[3] \end{bmatrix}$$

$$a^l = \begin{bmatrix} O[1] & O[2] & O[3] \end{bmatrix}$$

The O[1], O[2] and O[3] are the three outputs from the three neuron in the 2^{nd} layer. Each outputs from the neuron is computed by the MAC unit. The MAC unit multiplied the inputs with the respective weights and then accumulate all the multiplied output.

The neuron architecture in Figure 2.4 consists of an array of multipliers. There are hundreds of neurons in one layer which consists of hundreds of MAC units. With a large number of multipliers and adder blocks, there is an area overhead. It also leads to increased power dissipation (static power dissipation) by lowering the technology node.

The researchers have applied two ways of computing the MAC operation, Bit-Parallel Computing and Bit-Serial Computing. These two categories of works are

elaborated in the following two subsections.

2.2.1 Bit-Parallel Computing

Bit-Parallel architecture is the traditional method used for the multiply and adds operations. Figure 2.5 shows the MAC operation for two 4-bits input using the bit-parallel architecture. In this architecture, the input and weight are applied parallelly to the compute unit, X_i represents the input, and W_i represents the weight.

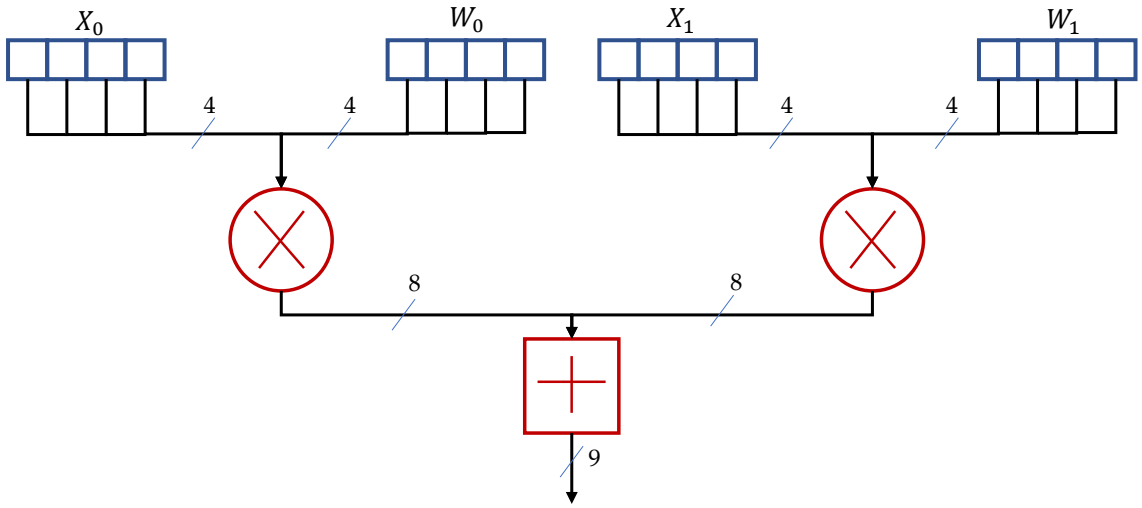


Figure 2.5: Multiply-accumulate operation using bit-parallel computing architecture

The advantage of such architecture is that all the bits apply simultaneously, so there is no problem on the latency front. However, this type of computing architecture is resource-hungry. For two 4-bit inputs and two 4-bit weights, it requires two 8-bit multipliers and one 9-bit adder. The challenge in such architecture is that as the bit precision increases, the hardware requirement increases quadratically. For example, in Figure 2.5 if the input and weight bits are doubled, then the hardware requirements for implementing the MAC operation will be four times the existing one. As the resources increases, the power consumption also increases proportionally.

2.2.2 Bit-Serial Computing

Researchers are exploring the bit-serial computing architectures to overcome the challenges proposed by the bit-parallel computing architecture. In these architectures, one of the inputs to the multiplier unit is serialized. It results in the decrements of the resources required. Figure 2.6 shows the MAC operation using the bit-serial computing for two 4-bits input and weight. As shown by serializing the weight input, the multiplier precision required is half required during bit-parallel architecture.

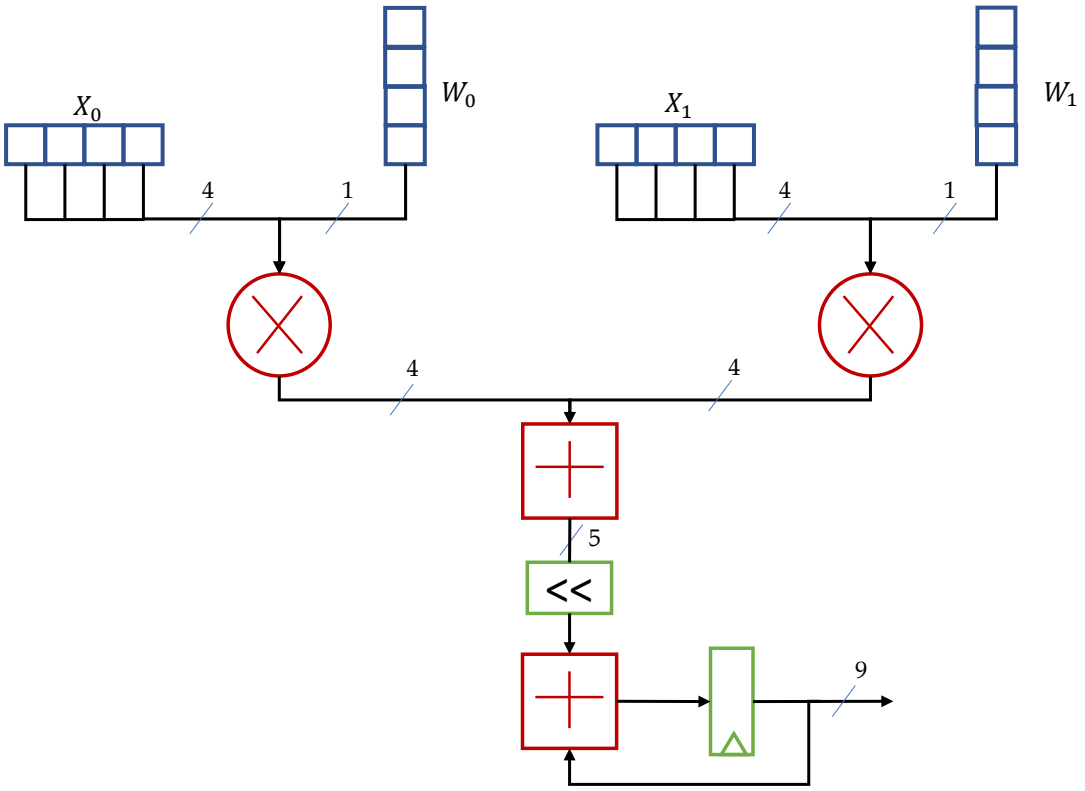


Figure 2.6: Multiply-accumulate operation using bit-serial computing architecture

In Figure 2.6 the 4-bit input X_0 and X_1 is multiplied by the 1-bit serialized weight W_0 and W_1 respectively. By serializing the weight input, the multiplier required is of 4-bit precision, which is of 8-bit precision in the case of bit-parallel computing as shown in Figure 2.5. After multiplication, the multiplier outputs are accumulated. In the case of bit-parallel computing for two 8-bit multiplied outputs, an 8-bit precision adder is required. Whereas in bit-serial computing, by serializing the weight input, we

will get four 4-bit multiplier outputs. The 4-bit multiplied output is summed using a 4-bit precision adder. For four 1-bit weights, we get four summed outputs of 5-bits each. The summed output is then required to be properly left-shifted and finally accumulate using an 8-bit precision adder. As shown in Figure 2.6, four iterations are there, so a feedback path is present on the 8-bit precision adder. The feedback has a register that saves the current output and accumulates it with the following output. In this case, after four iterations final output is generated.

The hardware resources utilized in an FPGA board is measured by the number of look up table (LUT) used. On the Zybo FPGA board, a single 8-bit multiplier and 4-bit multiplier requires 16 LUTs and 2 LUT's respectively. A 4-bit precision adder and an 8-bit precision adder require 4 LUT's and 8 LUT's respectively. The MAC operation for two 4-bit input and weight using bit parallel computing architecture as shown in Figure 2.5 requires two 8-bit precision multipliers and one 8-bit precision adder. For the same condition using bit-serial computing architecture as shown in Figure 2.6 requires two 4-bit precision multipliers, one 4-bit precision adder and an 8-bit precision adder. Overall the bit-serial computing architecture requires 20 LUTs and bit-parallel architecture requires 48 LUTs. The architecture designed using bit-serial computing is extremely resource efficient as compared to bit-parallel computing architecture. The power dissipation is also less in the bit-serial computing architecture as the resource utilization is good.

Stripes [22] shows that by using bit-serial computing, the network's performance can be increased without any accuracy loss in the computation. Tartan [23] also uses bit-serial computing units to improve precision flexibility. Both Stripes and Tartan have an additional area overhead to improve computation flexibility. Bit Fusion [24] uses the bit-level processing elements for decreasing the computation load and the communication between the blocks without any loss of accuracy.

The bit-serial computing reduces the resource requirement to some extent. However, there is still some need to modify the MAC unit on the logic architectural background for further reduction.

2.3 State-of-the-art Multiplier architectures

The most important block of the MAC unit considering the overall performance is the multiplier unit. To reduce the parameters of the MAC unit, the multiplier resources should be reduced. By reducing the parameters of the MAC unit, the overall network efficiency increases. There are various logic architectures of the multiplier proposed by the researchers. Using these different multiplier architectures, different configurations of the MAC unit are obtained in state-of-the-art architecture. The following subsection will tell about some of the vital logic architecture used for the multiplier architecture.

2.3.1 Array Multiplier

Array Multiplier is the most basic form of a multiplier. It is popularly known for its regular structure. There are two stages in the operation of the array multiplier. First is the partial product generation, which is carried out by simply performing the *AND* logic between the multiplier and the multiplicand bit. For an n-bit multiplication, n^2 *AND* gates are required. The generation of the partial product is performed in a parallel manner. The second stage involved the accumulation of partial products. The accumulation is carried out by half and full-adders. The accumulation is performed in a matrix-like fashioned arranged adders. For an n-bit multiplier and multiplicand $n \times (n-1)$ full adders are required. The architecture for the 4-bit multiplication is shown in Figure 2.7. The top row of the accumulation can be performed using half-adders and the rest with the full-adders as shown in Figure 2.7. Then it requiring $n-1$ half-adders and $(n-1)^2$ full-adders for the accumulation stage.

An efficient multiplication using an array-based multiplier is proposed in [25, 26]. The key feature of the array multiplier architecture is its simple design. The main disadvantage of such architecture is that its resource utilization is high. Due to the high resources, the power dissipation is also high. Also, the sum and carry from the adders process in a ripple fashion. The carry-in for an adder is coming from the adjacent full adder in the row and the sum from the above row. This makes the critical

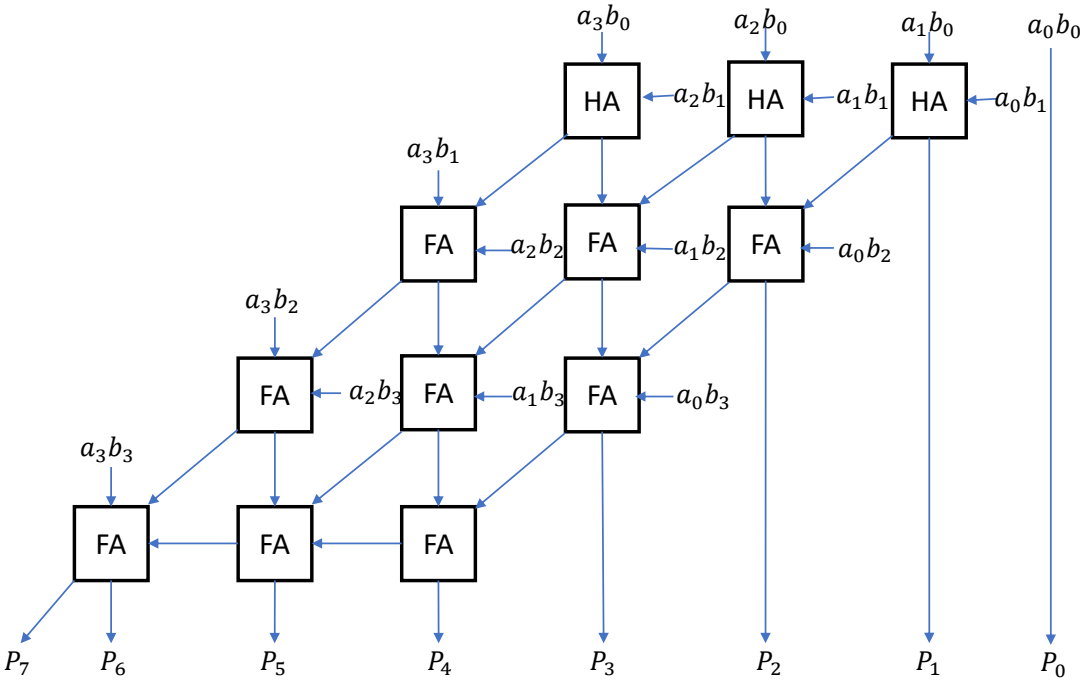


Figure 2.7: Block diagram of Array multiplier for 4×4 bit multiplication

path delay also high. Thus the architecture speed is also not competitive.

2.3.2 Wallace Tree Multiplier

To overcome the disadvantage of the speed in array multiplier, researchers come up with faster architecture known as Wallace Tree Multiplier [27]. The Wallace tree implies a three-step structure. First, the partial products are generated using the AND logic, and each partial product is associated with a weight depending on the bit position. Second, the reduction of the partial products is performed using a tree structure of adders. The tree structure has irregular interconnects between the adders to reduce the partial products into a two-row matrix. Lastly, summation of the two-row matrix to get the final output. The Wallace tree multiplier uses the log-depth tree network to reduce the partial products. The block-level architecture for 4-bit multiplication is shown in Figure 2.8.

The Wallace tree architecture is faster because its height is not linear. It is logarithmic in word size [28]. For an n-bit column height, the bits are divided into a

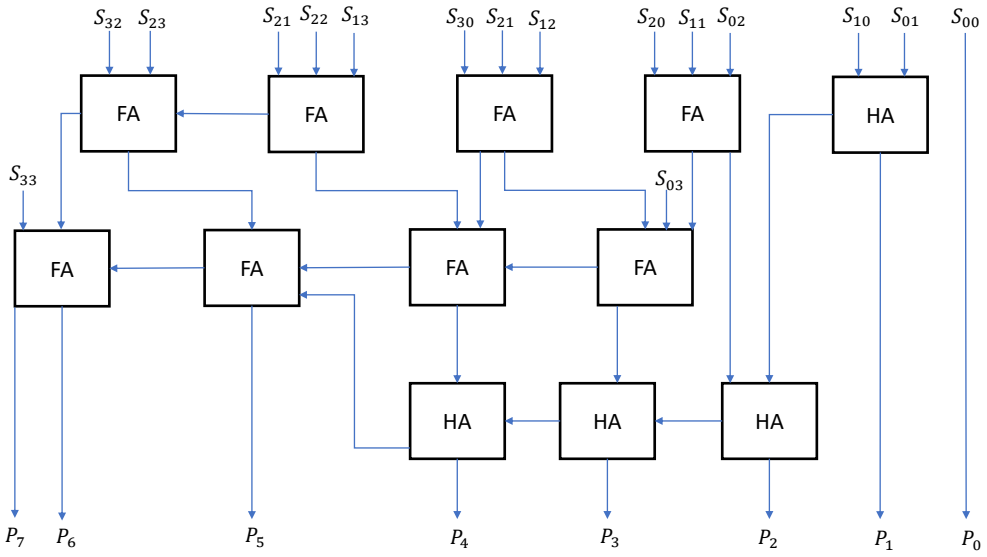


Figure 2.8: Block diagram of Wallace tree multiplier for 4×4 bit multiplication

group of three bits each. These groups are then reduced to two-bit, resulting in a new height of the column that is $2/3n$. The remaining bits are again divided into three-bit groups and reduce until the column height becomes two. The architecture in the second stage reduces the partial product at the rate of $log_{(3/2)}(N/2)$. However, the connections of the tree structure are irregular, which increases the design complexity of the architecture. As the design complexity increases, the overall area also increases. The architecture is generally avoided in low-power applications because it does not show any promising results on power consumption.

2.3.3 Shift and Add Multiplier

Shift and Add multiplier is equivalent to the multiplication performed using paper and pen [29]. It is a sequential operation. The block diagram of n-bits multiplication is shown in Figure 2.9. The algorithm is designed to traverse the bits of multiplier starting from right to left. Depending upon the bit, the control unit instructs the multiplicand, ALU, and product block. If the multiplier bit is logic '1', then the copy of the multiplicand bit is shifted and added with the ALU result. Otherwise, if the multiplier bit is logic '0', then the number of zeros shifted and added with the ALU result. The shifting operation is controlled depending upon which bit of the multiplier

is traversed. The multiplicand bit size is taken $2n$ bits to accommodate the shift operation properly. The actual size of the multiplicand is n bits only.

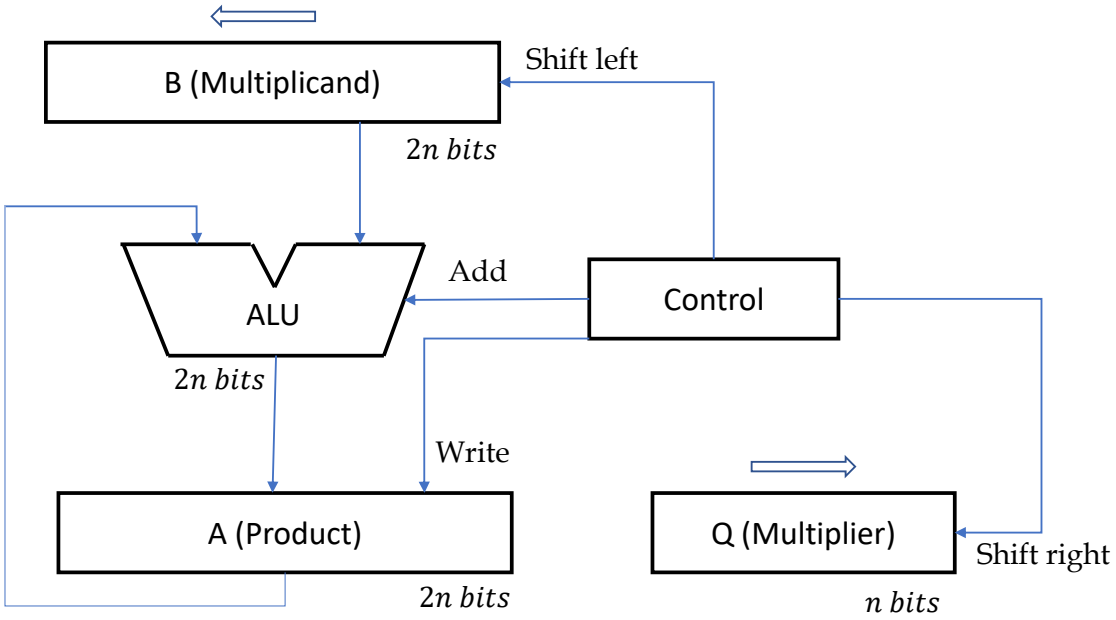


Figure 2.9: Block diagram of Shift and Add multiplier

The shift and add architecture has shown promising results in low resource utilization and low power [30]. Because it shows excellent results on the physical parameters, researchers have used this architecture in the CNN [31]. The challenge associate with this architecture is that it is not suitable for fast multiplication. Also, it can only perform the unsigned multiplication.

2.3.4 Vedic Multiplier

An ancient Indian Vedic mathematics-based Vedic algorithm is designed for multiplication in [32, 33]. In Vedic mathematics, there are 16 sutras and 13 up-sutras. With the help of these sutras, any mathematical problem can be solved. Out of the 16 sutras, the Urdhva-tiryagbhyam sutra is mainly used for the multiplication operation [34]. This sutra is also called vertical and crossbar algorithm architecture. Using vertical and crossbar algorithm, the 2-bit Vedic multiplier architecture is shown in Figure 2.10. The two numbers taken into account are $A = a_1a_0$ and $B = b_1b_0$. The

multiplication process starts with the vertical multiplication of a_0 and b_0, which gives the first bit (least significant bit) of the output. Then the cross multiplication between a_1-b_0 and a_0-b_1 is performed. These cross multiplication outputs are added. The sum is the second bit of the output, and the carry is added by the vertical multiplication of a_1 b_1. The final adder gives the last two bits of the output.

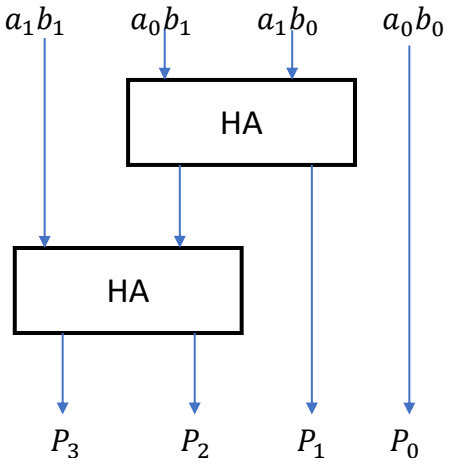


Figure 2.10: Block diagram of Vedic multiplier for 2×2 bit multiplication

A Vedic multiplier efficient way of developing the architecture for the higher bit precision uses the multiple lower bit precision architecture. For example, for 4-bit multiplication, four 2-bit Vedic multiplier and three 4-bit adders are required as shown in Figure 2.11. In this case, the 4-bit is divided into 2-bit groups and applied across the 2-bit Vedic multiplier.

The Vedic multiplier speed is good. Also, its delay increases slowly as the bit precision increases. However, due to the vertical and crossbar multiplications, the complexity of the circuit is increased. Further, it increases even more at higher precision, making it not an excellent choice to use.

2.3.5 Conventional IEEE Multiplier

IEEE specifies the standards and methods for floating and fixed-point arithmetic in the computer system. IEEE packages provide the optimized arithmetic functions which we can use by the operator of the operation, which is '×' for multiplication, '+'

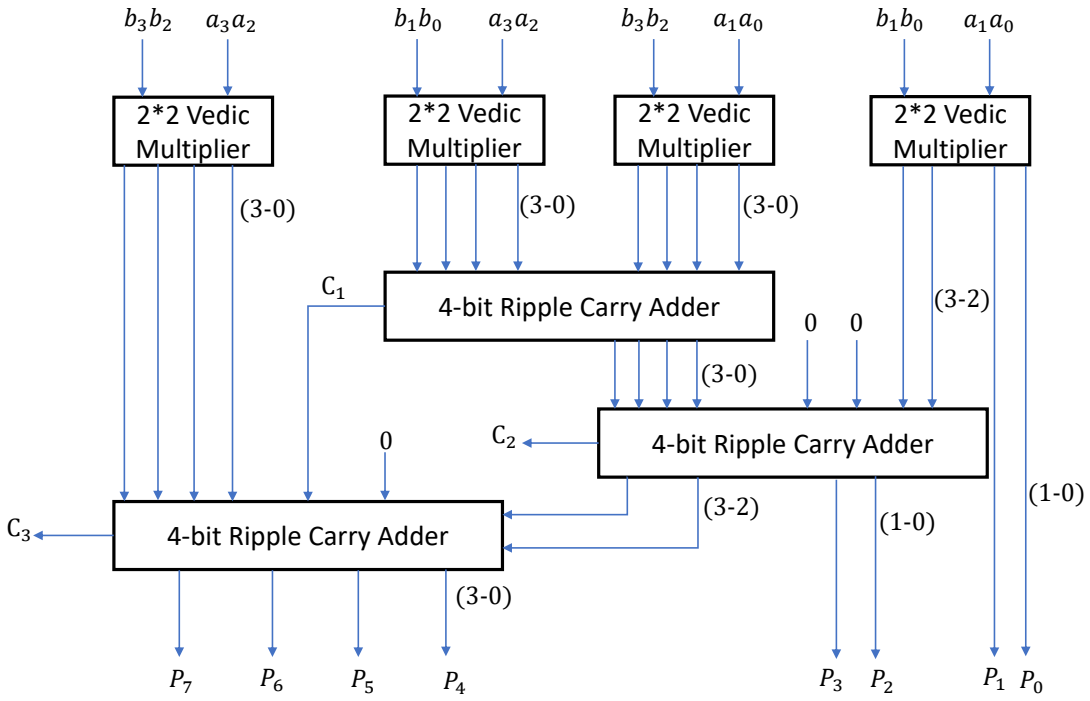


Figure 2.11: Block diagram of Vedic multiplier for 4×4 bit multiplication

for addition, and many more given in [35]. The electronic design automation (EDA) tools provide the intellectual property (IP), which can be directly used for arithmetic operations based on the IEEE standards. The numeric standard library package of VHDL/verilog is used. For multiplier in ASIC design it uses DSP based architecture whereas in FPGA it used the LUT based implementation.The different operations design is optimized for different physical parameters like resource utilization, area, power and delay.

2.4 Approximate Technique

To improve the performance by compromising the neural network's output accuracy, researchers have designed an approximate algorithm for multiplication and addition used in the MAC computational unit [26, 36–38]. These techniques are the quantized version of the well-defined multiplication algorithm. In [36] the author proposed an approximation algorithm on the Wallace tree multiplier to reduce the power consumption. In [26] approximate array-based multiplier is designed to improve cir-

cuit delay and power consumption. An approximate architecture that considers an approximate partial product accumulation tree is proposed in [37]. The authors also design multipliers using approximate logic compressors in [38]. Moreover, there is an improvement in the power and energy efficiency using these techniques. However, it is only limited to error-resilient applications only because of the reduction in accuracy of the approximate network.

2.5 Summary

Different neural networks such as CNN or ANN have shown propitious results in image recognition and classification application. With the increase of each layer's size, precision, and depth, there is an improvement in the network's learning capability and inference accuracy. This optimization, however, comes at the cost of increased computational complexity. In a hardware implementation, increasing computational complexity has an unfavorable impact on area and performance. Computation with higher bit precision (32-bit or 64-bit) is also expensive in terms of area and power [40]. Thus, the accurate computation with a reduction in hardware complexity (i.e., bit precision) and faster response without compromising accuracy is highly desirable [41]. Different architectures have been proposed to reduce the parameters like area, power and delay. The advantage and disadvantage of the various state-of-the-art multiplier architectures is shown in Table 2.1 However, there is a trade-off present in the parameters and their impact increases at the lower technology node. To overcome all these issues at lower technology nodes, we designed an efficient bit processing element

Table 2.1: Comparison of the state-of-the-art multiplier architectures

Multiplier Technique	**Advantages**	**Disadvantages**
Array Multiplier [25, 26]	Design Complexity is low	High power, delay and resource utilization
Wallace Tree Multiplier [27]	High speed /throughput	High power and area utilization
Shift and Add Multiplier [30, 31]	Power efficient	Low throughput and only unsigned multiplication possible
Vedic Multiplier [33, 39]	High speed	Design complexity is high and high power
Approximate Multiplier [36–38]	Power and resource efficient	Reduction in accuracy

and designed a high precision architecture of a MAC unit using bitcell. The proposed architecture has the benefits of efficient power, area and delay of the circuit.

Chapter 3

Compute-efficient multiply-accumulate unit

Computational power is the most critical parameter in today's application environment. Due to the extensive use of computational operation the neural network applications are power-consuming. The computational operation in the neural network is mainly performed using the MAC unit. The MAC unit is the primary building block of neural network architecture. It is the most repetitive unit in the computation. It is shown to consume 90% of the overall computational power of the neural network [13]. In a neural network, the critical physical parameters like area, power and delay are primarily dependent on the MAC unit's performance. An optimized MAC in terms of area, power and delay is required to optimize the physical performance parameters.

The bit-serial computing is efficient in terms of system performance. It reduces the hardware footprint to a reasonable extent. Therefore we have designed the MAC using the bit-level processing technique. To make a more efficient architecture, we bypass the multiplier with a bitcell based multiplier and accumulate architecture. The architecture is explained in detail in the following sections.

3.1 Bitcell Architecture

The bitcell is a unit in which a single-bit operation is performed. The bitcell is designed considering the constraint that the architecture takes minimum resources. Bitcell consists of a 1-bit memory cell for storing the weight constants, *XNOR* logic gate for bit-wise processing, and a half-adder as shown in Figure 3.1.

Two operations are being performed in the bitcell, one is *XNOR*, and the other is the addition. Both of these operational blocks are not active simultaneously. First, the *XNOR* operation is performed between input (x) and weight (w), and then the output of the *XNOR* gate is added with the carry input (Carry in) using half adder. The single-bit adder's output, the sum, and carry are the two outputs of the bitcell unit. The *XNOR* is chosen because of its inverting property, which complements the other input when one input is logic '0'. In this bitcell architecture, the task of *XNOR* is to complement the weight bit if the input bit is logic '0'and passes the weight bit if the input bit is logic '1'.

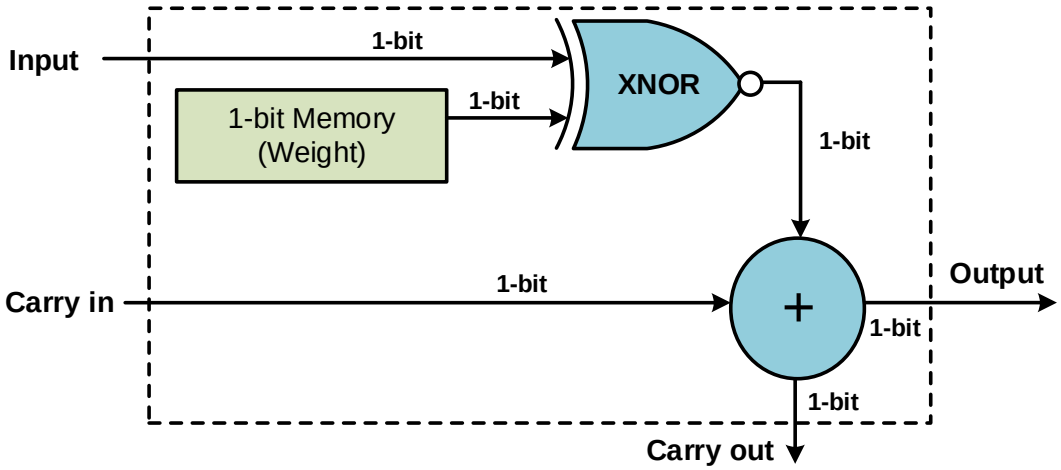


Figure 3.1: Efficient design architecture of bitcell used for each single bit calculation

The bitcell architecture is optimized in terms of resources. For the complete MAC implementation, we will use the combination of the bitcells in place of the multiplier architecture. The combination of the bitcells will perform the necessary multiplication operation with minimal hardware possible.

3.2 Bitcell Optimization using Power Gating

The technology node is scaling at a rapid pace due to several advantages at lower node technology. Nevertheless, this brings some challenges as well. The biggest problem for a designer is the increase of the leakage current at the lower technology node. The increase in the leakage current creates many problems in the circuit. It increases the static power dissipation to a great extent. The variation in the physical parameters is observed with the technology scaling. Figure 3.2 shows the variation of the two critical physical parameters, delay and *ON* current of the inverter concerning the different technology nodes [9]. The technology scaling impacts the technology parameters like velocity saturation (V_{sat}), affecting the device current and delay in the circuit. The parameter delay shows significant improvement on scaling down from higher technology node to lower technology node. As we scale down the technology beyond *90nm*, the *ON* current starts increasing drastically, leading to a higher static power dissipation at the lower technology node. The area, dynamic power and delay are decreased at the lower technology node, but the static power dissipation increases at a higher rate.

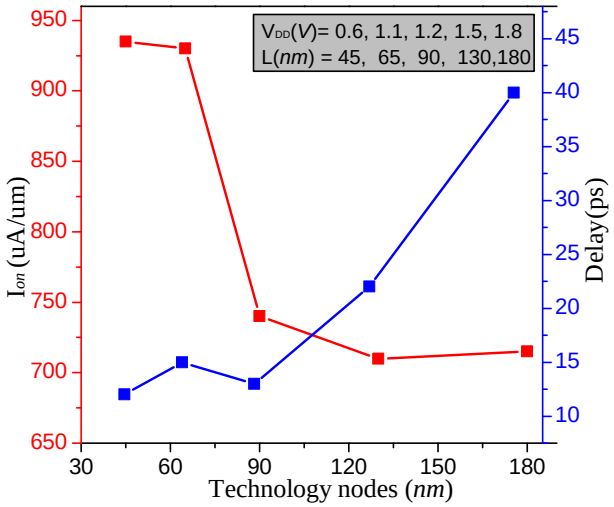


Figure 3.2: Effect of technology node on current and delay

The leakage current is generated from the computational block, which is in the idle or off state. The bitcell logic block works in a sequential mode. There are two

computational blocks, *XNOR* and adder, in the bitcell. Out of which, the adder block input is dependent on the output of the *XNOR* block. The adder block is in an idle state till the time it gets input from the *XNOR* block. Also, during one-half of the cycle, both the computational block are in idle or hold state. This is when the leakage current is dominant in the circuit, which eventually increases the static power. To reduce that static power, we have implemented the power-efficient bitcell using the power gating technique.

Power gating is an effective technique that is used in low-power designs [42]. In this technique, the current to the blocks is shut off, which is not in use. This technique is effective where computational blocks are in an off state for a significant time during operations. The bitcell is designed such that the power gating technique can be implemented.

In the power gating technique, extra metal-oxide-semiconductor field-effect transistor (MOS) are inserted in the design. The MOS acts as a switch and bypasses the supply voltage from the computational block when idle. With bypassing the supply, there is minimal leakage current which results in low static power dissipation. Based on simulation results and merit, the power gate size should be around 3× large as compared to its standard size to maintain similar performance [43]. Based on the position of the MOS, there are two types of design configuration which are used. These configurations are as follows:

1. Fine-Grain Power Gating Technique: In this technique, the power gated MOS is added for each cell that needs to be shut off in an idle state [44]. This configuration is not preferred most of the time. The problem with this configuration is that it increases the complexity of the design. Also, by adding MOS for every cell, there is an area overhead.

2. Coarse-Grain Power Gating Technique: The MOS is added for a whole core/block instead of each cell in this technique [45]. So when the MOS is in an off state, the whole block is turned off. The area overhead is significantly less compared to the fine-grain technique. Also, bypassing the whole block is

much easier than bypassing individual cells. There are two configurations in the coarse-grain technique as well, which are mentioned below.

(a) Column-based coarse-grain power gating technique - In this technique, the MOS is inserted for the whole column in a grid-like fashion [46]. This technique is mostly used when we need to retain the data in the computational block during the sleep mode. The MOS is closer to the design block in this design, so the IR drop is less, and the virtual VDD routing length is also small.

(b) Ring-based coarse-grain power gating technique - In this technique, as the name suggested, the MOS are inserted in a ring fashion around the design [43]. This technique is used when there is no constraint to retaining the data in sleep mode and when we do not want to interrupt the original design. The IR drop is higher in this configuration, but it is less complex than the column-based technique.

To reduce that static power, we have implemented the power-efficient bitcell using the power gating technique as shown in Figure 3.3

We have used the column-based coarse-grain power gating technique because the area overhead is less compared to the fine-grain technique. Also, the proposed architecture of MAC consists of a combination of bitcell architecture in column fashion, so the column-based power gating technique is suitable for the design. We have used a single set of only 4 MOS and delay elements for a complete MAC unit that contain multiple bitcells, and hence, there is not much area overhead. As the adder operation depends upon the *XNOR* output (y_{out}), a delay element is included in the path from the clock to the two MOS connected with the adder block. The delay of the path is adjusted with the delay of obtaining the output (y_{out}) from the *XNOR* block. The transistors used for the power-gating technique are of higher gate width so that the functionality of the circuit is not affected. The column-based coarse grain technique reduces the circuit complexity, provides better efficiency and moderate switching time.

In power gated bitcell, we used four MOS transistors to isolate the current path

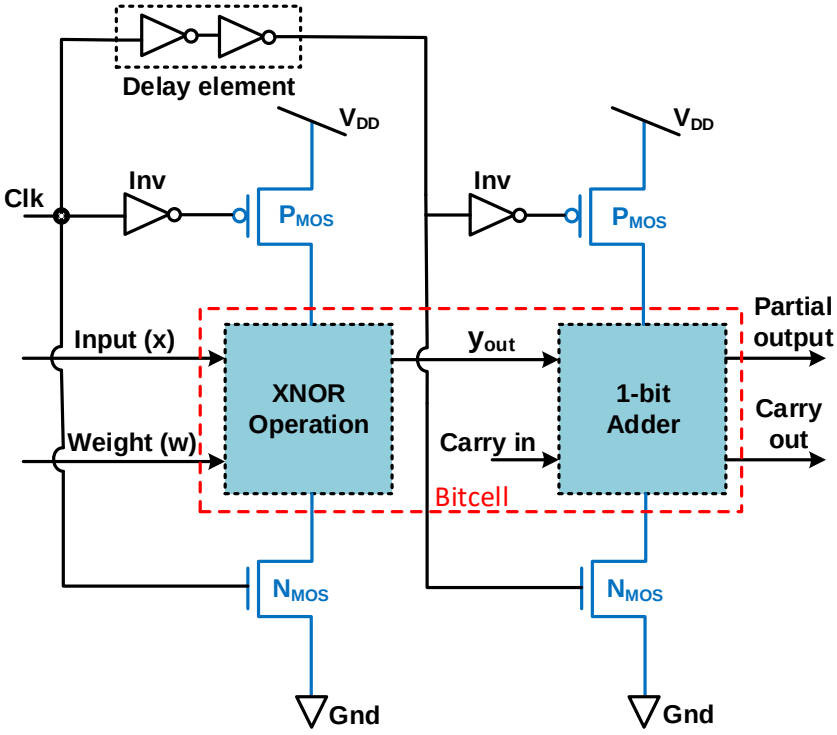


Figure 3.3: Power-efficient bitcell logic architecture with coarse-grain PG technique

between supply (V_{DD}) and ground (Gnd). When the clock (Clk) is at a positive level, all the MOS are in the saturation region and act as a closed switch establishing the connection path between V_{DD} and Gnd. In the positive half of the clock, both the operations *XNOR* and addition are performed. After that, they need to hold the data. When the clock is at a negative level, at logic '0', all the MOS are in the cutoff region and act as an open switch, thus isolating the voltage source connection from the logic circuit. The *XNOR* logic gate dissipates more static power when it is *ON*. To decrease the static power dissipation for the positive half cycle *XNOR* gate is power gated, i.e. when the *XNOR* block is at the idle (hold) state.

3.3 Bitcell Performance Parameters Analysis

The bitcell is designed for single-bit processing using basic logic components, i.e., *XNOR* gate and half-adder. First we design a logic without power gating technique and performance parameters is observed for both *180nm* and *45nm* technology node. Then

Table 3.1: Performance parameter metrics for power gated and non-power gated bitcell at both 180nm & 45nm technology node.

Performance	**for 180nm**		**for 45nm**	
parameters	non-PG	with PG	non-PG	with PG
Area (μm^2)	55.18	136.60	11.55	48.40
Dy. Power (μW)	1.36	1.39	0.58	0.63
St. Power (nW)	1.04	0.71	59.68	36.23
Critical Path Delay (ns)	2.50	2.78	2.16	2.33

we design the bitcell with the power gating technique and validate the performance at both technology nodes. The important physical parameters for power gated bitcell and non-power gated bitcell shown in the Table 3.1. We observed 31.24% and 39.19% less static power in power gated bitcell as compared with non-power gated bitcell at *180nm* and *45nm* technology node respectively. In power gated bitcell, the critical path delay is 11.27% and 7.87% higher as compared with the non-power gated bitcell at *180nm* and *45nm* technology node, respectively. From the observation to the downturn of the increasing static power dissipation, we have taken advantage of the power gating (PG) technique in the bitcell unit and the MAC design implementation.

3.4 N-bit Precision MAC using Bitcell

The architecture for the MAC is designed such that it takes minimum resources and minimum power. The primary function of MAC is to multiply two inputs and accumulate the product $i.e.(A = A+B*C)$. By using the bitcell based MAC technique, variable precision MAC can be implemented. For designing an N-bit precision MAC, N number of bitcells are required. The N bitcells are connected in parallel, following the shifting and accumulation stage. For explanation, 8-bit precision MAC is designed

and compared with the state-of-the-art-architectures. The following sections explain the architecture design and working in detail.

3.4.1 8-bit Precision MAC architecture

The MAC unit is made of an array of bitcell units. The block design of logic architecture for the proposed 8-bit precision MAC unit is shown in Figure 3.4. The proposed 8-bit MAC consists of eight bitcells in an array, followed by the left shifting using shift registers and then sequential accumulation after each operation. The input features are transmitted to all the eight bitcells. The bit transmission is serially starting from the least significant bit (LSB) to the most significant bit (MSB).

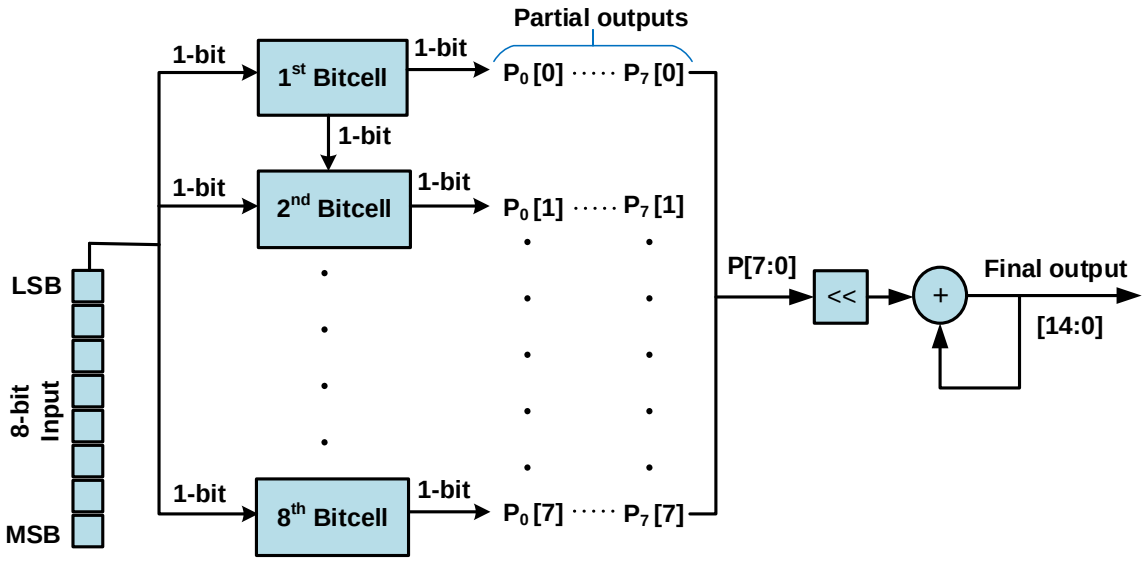


Figure 3.4: Design architecture of 8-bit precision multiply-accumulate unit using bit-cell architecture

3.4.2 Working of the 8-bit precision MAC

Each bitcell takes the input bit, processes it with the weight stored in that particular bitcell, and gives the 1-bit carry output and 1-bit sum output. The output carry of the bitcell is given as the input carry to the subsequent bitcell. Note that the input carry for the first bitcell is the complement of the input bit. The eight bitcells

produce an 8-bit output called a partial product for a single-bit input. For 8-bit input, there will be eight such partial products. These partial products are left shifted depending upon in which input bit iteration the partial product is generated.The left shift computation on partial products block for 8-bit bitcell based MAC is shown in Figure 3.5 The partial product generated in the N^{th} iteration is left shifted by N-1. All the shifted partial products are then accumulated to give the final output of the multiplier. For the proper accumulation of the signed partial product, the MSB of each partial product is copied to the maximum output bit before adding the shifted partial products.

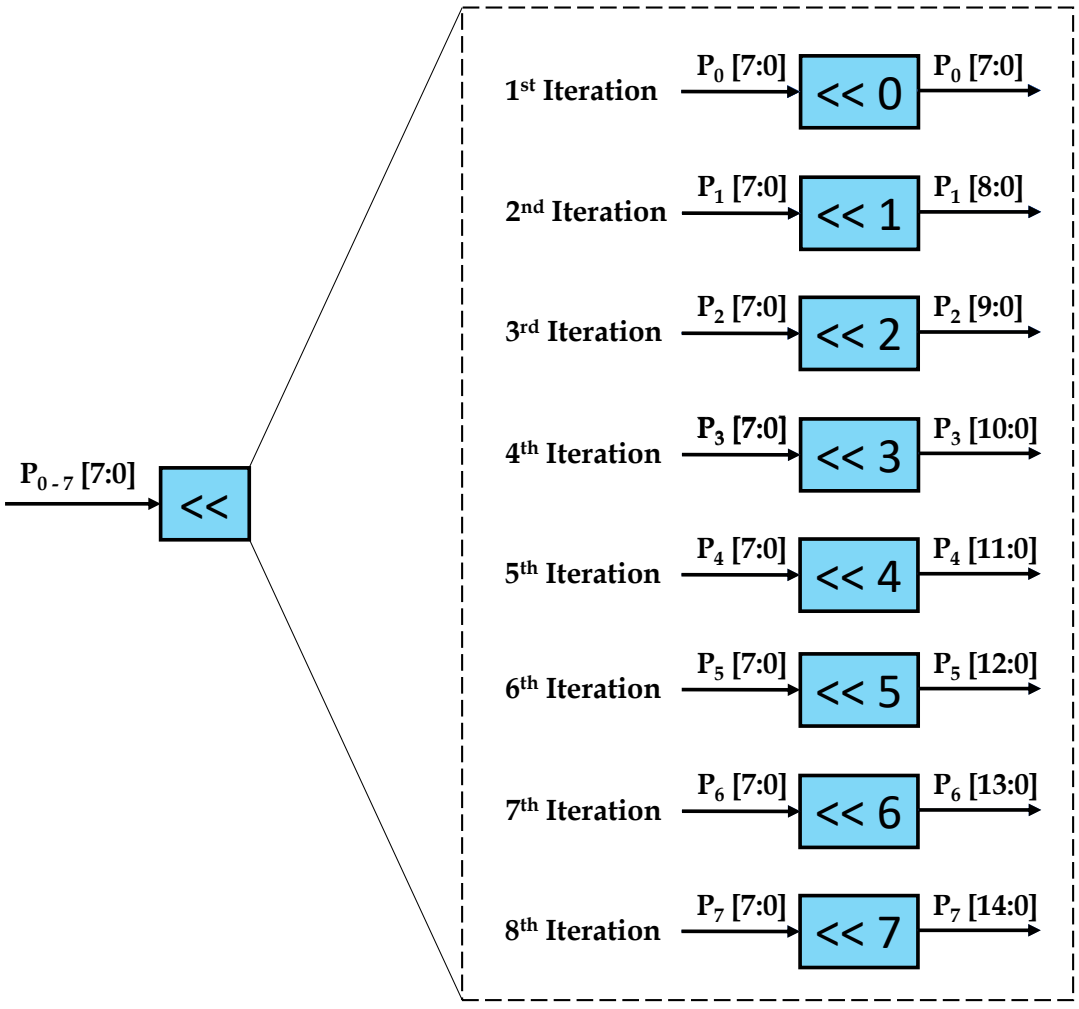


Figure 3.5: Shift operation computation in 8-bit bitcell based MAC architecture

The weight stored in the register is in the 2's complement signed format, and

the input bit is encoded based on positive and negative positional values to a binary-weighted signed number. The generated output by the array of bitcell is also in the 2's complement signed format. The binary '0'is encoded as negative of the positional value of that bit, and binary '1'is encoded as positive of the positional value of that bit. For example a 8-bit input '0011 0011'represents '$-2^7-2^6+2^5+2^4-2^3-2^2+2^1+2^0 = -153$'

The logic architecture and computation are elaborated through an example of the multiplication of 4-bit input (4'b0101 or -5) with 4-bit weight (4'b0101 or +5). For a 4-bit multiplication, four bitcells will be there in an array. In the first iteration, the LSB of the input i.e. '1'b1'is given to the four bitcells in which 1-bit weight is stored in each bitcell. The *XNOR* operation is performed on the input and the weight bit that gives a single bit output, which is added with the initial carry. For the first bit cell, the initial carry is the complement of the input bit (1'b1) i.e. 1'b0, and for the remaining three bitcells, the initial carry is the output carry of the previous bitcell. The four bitcells provide a 4-bit partial product (4'b0101) for the first input bit that is LSB. Similarly, for the remaining three input bits, three 4-bit partial products are generated that are '4'b1011', '4'b0101', and '4'b1011', respectively. The first partial product is left-shifted by 0. The second partial product is left-shifted by 1, and so on. All the shifted partial products MSB is being copied till the 7th bit, and then they all are added to give the final result that is in this case ''7'b1100111 or -25''.

As we have designed the MAC unit for 8-bit precision, the real-time operation simulation waveform for the data computation with each iteration is shown in Figure 3.6. The 8-bit input taken is '8'b11001001 or +147d'and the weight consider is '8'b00000100 or +4d'. The output of the multiplication is '15'b000001001001100 or +588d'.

The waveform in the consolidated form is shown in Figure 3.7. There are eight iterations for the 8-bit input. In each iteration, the input bit goes to the eight bitcells, and partial products are generated. As we can see in the waveform when the input bit is logic '1', the bitcell passes the weight, and when the input bit is logic '0', the negative of the weight bit is passed. The partial product is left-shifted depending on which iteration it is generated. The left-shifted output is named as 'Shifted output'

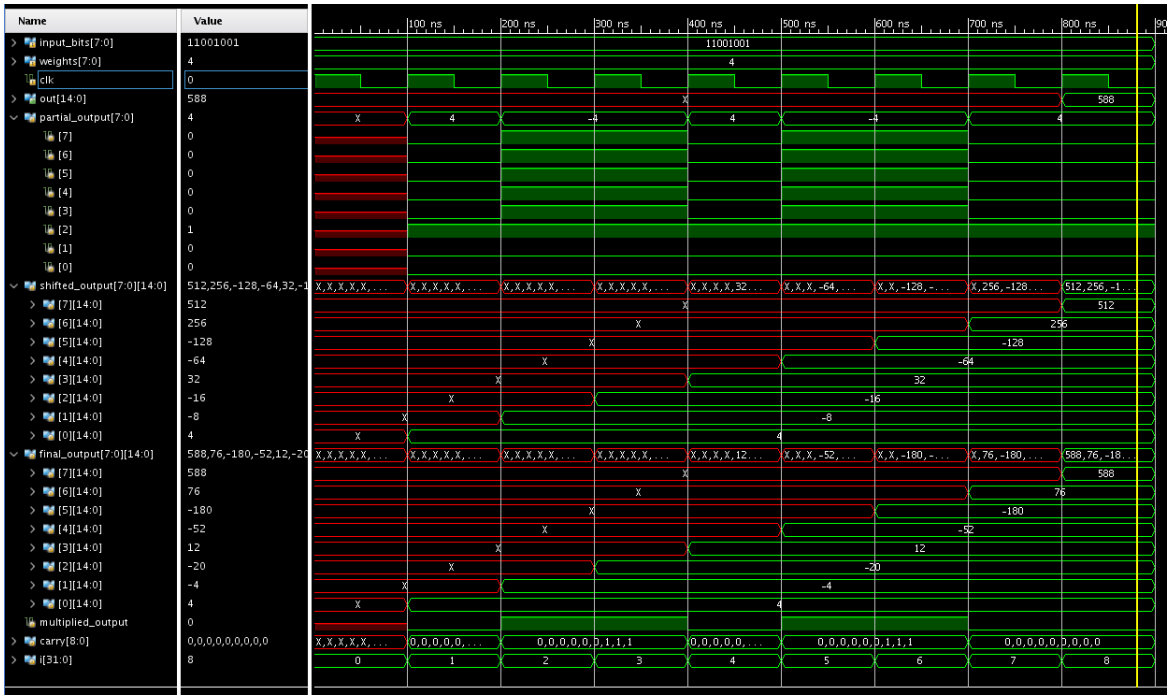


Figure 3.6: Simulation waveform for data computation with each iteration approaching the desired output. Simplified calculation is elaborated for decimal values with input=+147d and weight=+4d

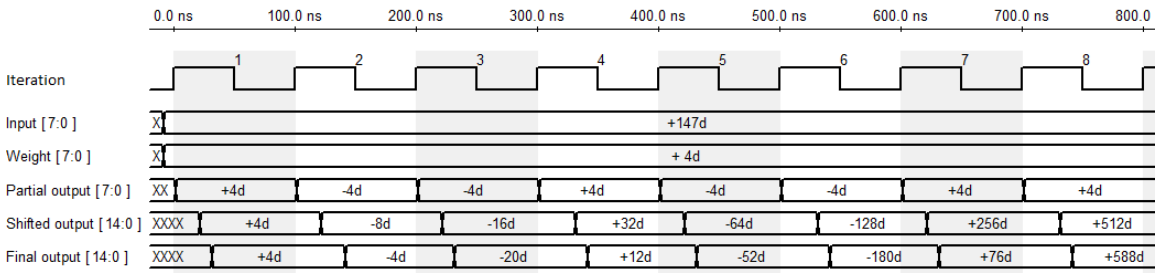


Figure 3.7: Consolidated simulation waveform for data computation with each iteration approaching the desired output.

in the waveform. At last, all the shifted outputs are accumulated and stored in 'Final output'.

The total time taken to achieve the final output is given as

$$t = T_{clk} \times n \tag{3.1}$$

$$where - T_{clk} > T_{critical} \tag{3.2}$$

Here,

t = total time requires for n-bit computation

n = bit-precision of input

T_{clk} = time period of clock

$T_{critical}$ = critical path time delay

Chapter 4

Convolutional Neural Network Implementation

In recent years the CNN is a widely used neural network. For image processing and recognition tasks, CNN is always the first choice. CNN has shown promising results in terms of accuracy and speed. Due to the increase in CNN's applications and computational capability, the need for a low-power system has increased exponentially. This necessity increases the research on hardware acceleration. Hardware Acceleration can be interpreted in different ways. In the case of CNN, the critical thing is to decrease the power consumption and resource utilization by sustaining the real-time accuracy and speed. The basic block diagram of the hardware accelerator is shown in Figure 4.1. The processing element is the main computation block. In this block, the MAC operation is performed. There are on-chip memories used to store the input bit, weight bit, partial results, and the final output results. The control unit generates the control signals to operate synchronously.

There are many architectures of CNN which have been developed over these years. Some of the popular architectures are LeNet-300-100 [47], AlexNet [18], VGGNet [48], GoggleNet [49] and ResNet [50]. Each architecture differs in the number of layers, size of the layer, and kernel size used. We have used the LeNet architecture for the performance evaluation. The explanation of the architecture and the performance analysis reports are discussed in the following sections.

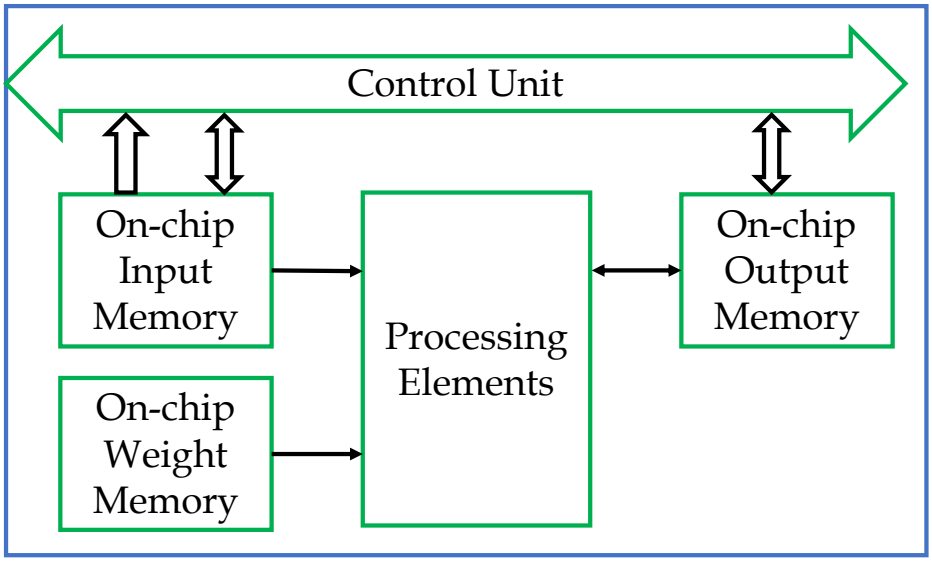


Figure 4.1: Block Diagram of CNN Accelerator Architecture

4.1 LeNet Architecture

The LeNet-5 architecture was the first architecture of CNN that successfully recognized the handwritten number of MNIST datasets. The limitation of the fully connected neural network was that it treats every pixel as an input and then processes it with the AF, which increases the computational burden. LeNet architecture has taken advantage of the fact that adjacent pixels are co-related, and features are distributed in the image. LeNet has changed CNN learning and shows that learning with shareable parameters is most effective. It reduces the number of parameters and gives good accuracy as well.

The LeNet-5 architecture has seven layers, excluding the input layer. Two layers are convolution layers, two are pooling layers, also known as subsampling layers, and three are fully connected layers. The block diagram of LeNet architecture is shown in Figure 4.2. The first layer is the convolution layer. It receives the input of image size 32×32 pixels. The filter size used is 5×5, and the depth of the filter is 6. The stride size is one with zero paddings. The size of output from this layer is 28×28×6 (32-5+1 = 28). The second layer is the pooling layer. The output from the first convolution layer acts as an input to this layer. In this layer, six filters of size 2×2 are used with

a stride of two. As the filter size and stride are the same, there is a non-overlapping sub-sampling operation, due to which the output size from the second layer is half of the outsize from the first layer. The output size from the pooling layer is 14×14×6.

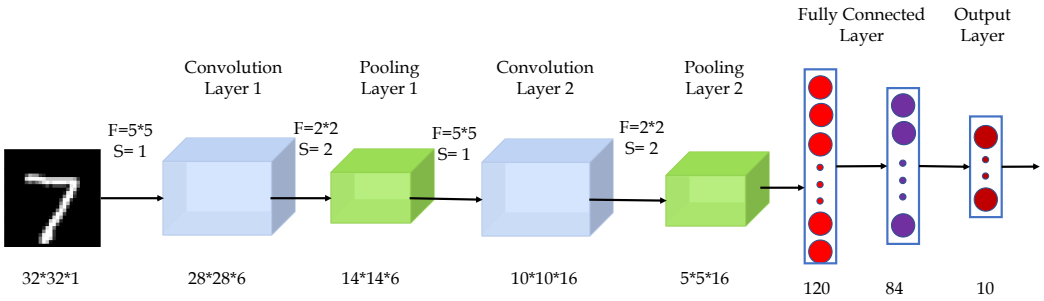


Figure 4.2: Block Diagram of LeNet Architecture

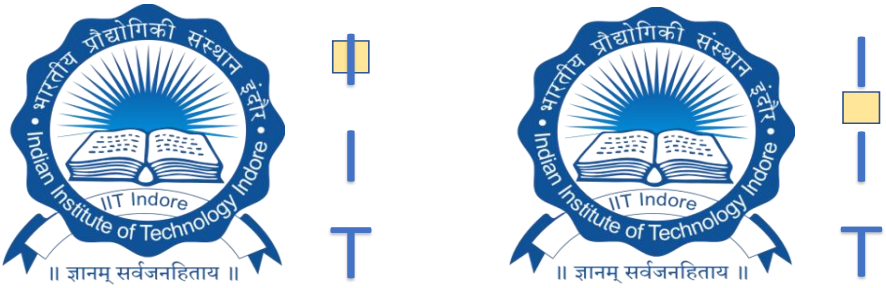
The third layer is again a convolution layer in which sixteen filters of size 5×5 are used with a stride of one. The output matrix size from this layer is 10×10×16. The fourth layer is the pooling layer that used the same number of filters with the same filter size as the second layer. The output from the fourth layer is 5×5×16. The fifth layer is the fully connected layer. All the 400 input features are connected with 120 output nodes in this layer. The sixth layer is also a fully connected layer that connects the 120 input nodes to 84 output nodes. The seventh layer is the final layer of the architecture, called as output layer. Ten output nodes correspond to the ten different digit values from 0 to 9. The softmax AF is used in this layer which gives the probability of each node value.

The feature extraction and classification in CNN are done by combining multiple computation layers. These layers form the basic building block of each CNN. The four types of layers used in the architecture are explained in the following subsections.

4.1.1 Convolution Layer

The convolution layer is the main computation layer in the CNN architecture. The convolution operation is the main computational operation in CNN. Before understanding the operation, we should know some basic definitions of image processing. The input image can be defined in mathematical values by a matrix of pixel values.

When the input is a colored image, the pixel has three channels named Red, Blue, and Green. In the case of a grayscale image, there is only one channel. There is a matrix of trainable weights, which is called filter or kernel. The filter is used to extract the features from the image. In the convolution layer, the input matrix is convolved with the filters. The filters are slid over the whole input and perform the matrix-matrix multiplication. It can be visualized as shown in Figure 4.3. The yellow color box is the filter. It will start shining the light from the top left of the image and slide over the image horizontally. It will then come vertically downwards and again slide over the image horizontally. It will do so till the whole image is covered. The convolution occurs in the same way between the image and the filter. The convolution operation is done to extract the features of the image. If the feature of the input match with the filter, the output matrix mathematical value is high. As in Figure 4.3(a) the feature is matching with the filter. However, in Figure 4.3(b), there will be no feature extracted, so the output matrix mathematical value will be low, like close to zero.



(a) Input feature match with the filter

(b) Input feature not match with the filter

Figure 4.3: Visualization of the filter sliding over the image

In the convolution operation, the filter slid over the image and performed convolution. It occurs for each combination of filter and input part defined by the size of the filter and the sliding number. A sliding number called stride size defines the space between two samples of the input image.

Given an image matrix of size $n \times n$

$$I = \begin{bmatrix} I_{11} & I_{12} & \dots & I_{1n} \\ \vdots & \vdots & \dots & \vdots \\ I_{n1} & I_{n2} & \dots & I_{nn} \end{bmatrix}$$

and a filter of size $f \times f$

$$F = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1f} \\ \vdots & \vdots & \dots & \vdots \\ w_{f1} & w_{f2} & \dots & w_{ff} \end{bmatrix}$$

the convolution operation applied on input and filter is given as

$$I * F = O_{ab} = \sum_{q=1}^{f} \sum_{i=1}^{f} w_{qi} \times I_{(a+q)(b+i)} \tag{4.1}$$

Where O, I, w are output matrix, input matrix, and filter weights, respectively. The equation can be better understood with an example of convolution as shown in Figure 4.4. In this example, the input image matrix is of size 3×3, and the filter size is 2×2. With stride one, the output matrix will be of size 2×2.

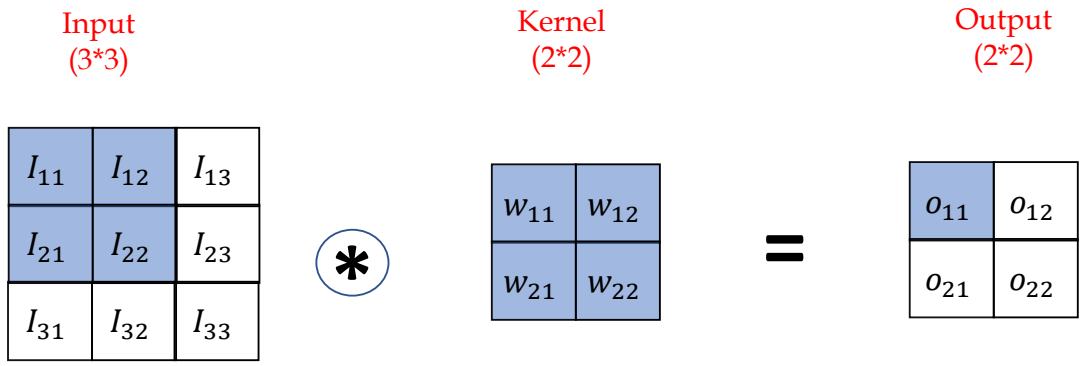


Figure 4.4: Convolution operation of 3×3 input matrix with the filter of size 2×2

On expanding the equation 4.1 for this example, the output matrix value will be given as

$$o_{11} = I_{11} \times w_{11} + I_{12} \times w_{12} + I_{21} \times w_{21} + I_{22} \times w_{22} \tag{4.2}$$

$$o_{12} = I_{12} \times w_{11} + I_{13} \times w_{12} + I_{22} \times w_{23} + I_{22} \times w_{22} \tag{4.3}$$

$$o_{21} = I_{21} \times w_{11} + I_{22} \times w_{12} + I_{31} \times w_{21} + I_{32} \times w_{22} \tag{4.4}$$

$$o_{22} = I_{22} \times w_{11} + I_{23} \times w_{12} + I_{32} \times w_{21} + I_{33} \times w_{22} \tag{4.5}$$

The input and weights are multiplied in the above equation, and all the multiplied products are accumulated. The MAC unit in the hardware carries out this operation. There are hundreds of convolution operations depending on the size. Thus by optimizing the MAC operations, there will be a significant effect on the complete architecture parameters.

There are two convolutions layers used in the LeNet architecture. Both layers specification, trainable parameters and connections are shown in the Figure 4.5 and Figure 4.6

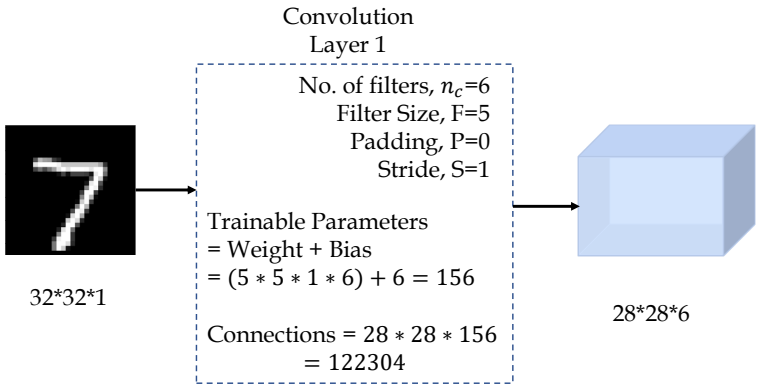


Figure 4.5: Specification of first convolution layer in the LeNet architecture

4.1.2 Pooling Layer

The pooling layer is mainly used between two convolution layers. The pooling layer is also called the sub-sampling layer. It reduces the feature map size by using filters. In pooling layers, the filter size is smaller as compared with the convolution

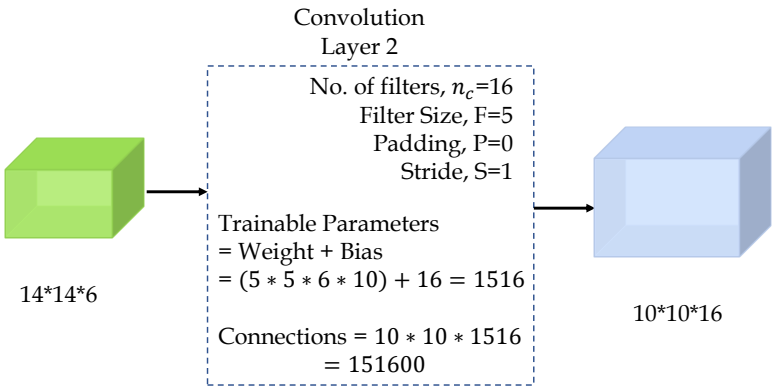


Figure 4.6: Specification of second convolution layer in the LeNet architecture

layer. The convolution layer detects the feature, and then the pooling layers decrease the size of the feature map by adding some non-linearity.

The pooling filter works in the same fashion as the filters do in the convolution layer. The filter slide over the image depending on the stride value. Pooling operation is performed on each part of the image selected by the filter by sliding. Generally, two types of pooling operations are used.

1. Average Pooling: Average pooling calculates the average for each portion of the image. An example of average pooling operation is shown in Figure 4.7. In this example, the average pooling operation is applied on 4×4 image part with the filter size of 2×2 and stride two. The different color indicates the image part selected by the filter size and stride. For each part, the average of the values is calculated.

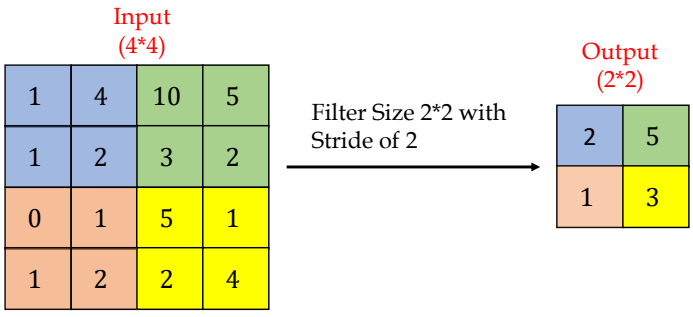


Figure 4.7: Example of Average Pooling Operation

2. Max Pooling: In max pooling operation for each part, the maximum value is selected. An example of max-pooling operation is shown in Figure 4.8. It is the most preferred pooling operation used nowadays. The advantage of this operation is that it keeps the most relevant information from an image part. The filter size is smaller because with a large filter size, probability of losing the important information is high.

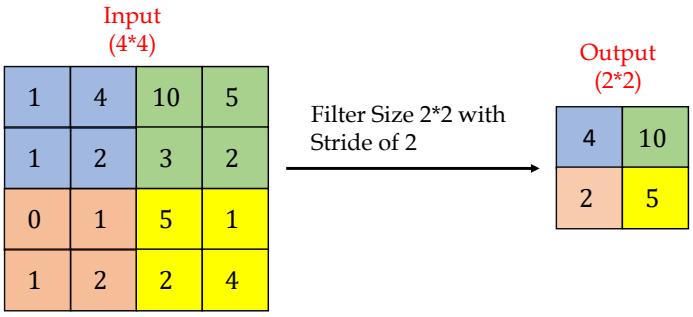


Figure 4.8: Example of Max Pooling Operation

There are two pooling layers used in the LeNet architecture. Both layers specification, trainable parameters and connections are shown in the Figure 4.9 and Figure 4.10

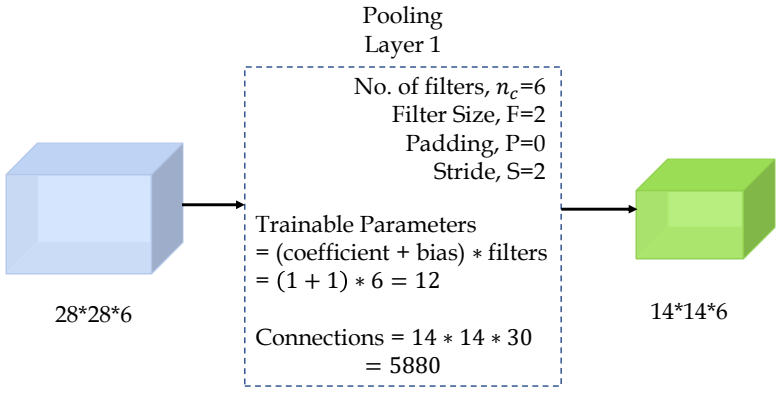


Figure 4.9: Specification of first pooling layer in the LeNet architecture

The stride value in the pooling operation is primarily selected as same as filter size. With the same size, there is no overlapping between the image parts. There are several advantages of using the pooling layer. As we go deeper into the layers, it decreases the feature map size without losing important information. The reduction

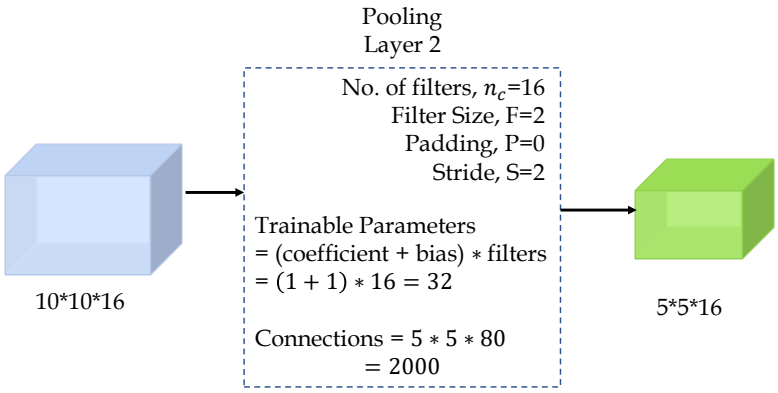


Figure 4.10: Specification of second pooling layer in the LeNet architecture

of size reduces the trainable parameters and also the memory to hold it. It shows an effect on the training time as well.

4.1.3 Fully Connected Layers

There are two main tasks in the CNN network, feature detection, and classification. The convolution layer detects the features, and the fully connected network takes the feature extracted and classifies them into different classes. The fully connected layers are used in the last layers of the CNN architecture. In fully connected layers, all the neurons are connected to the other neurons from the previous layer as shown in Figure 4.11.

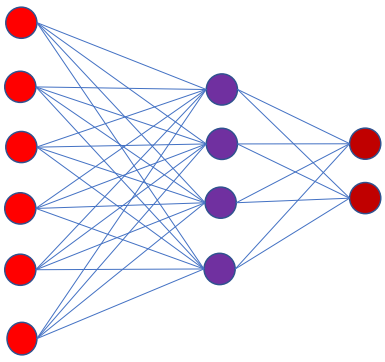


Figure 4.11: Fully Connected Neural Network

Each neuron is doing the basic operation which is MAC followed by the activation

function. Mathematically the each layer operation is defined as

$$y_i = f(\sum(W_i X_i + b)) \tag{4.6}$$

where,

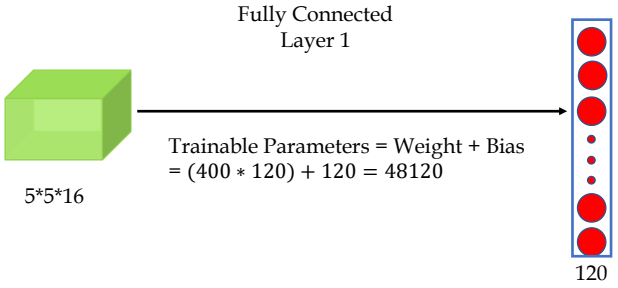
X_i - is the input vector with dimension [n,1] where n is the number of neurons in the previous layer

W_i - is the weight matrix with size [n,m], where m is the number of neurons in the current layer

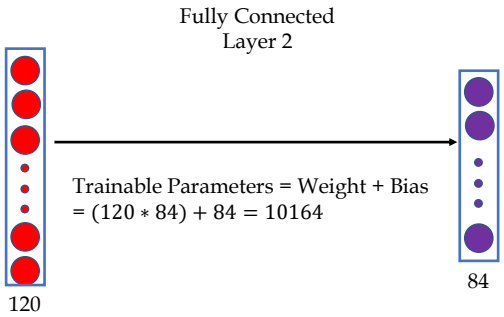
b - is the bias vector with size [n,1]

f - is the activation function

The details on the activation function are in the following subsection. There are two fully connected layers used in the LeNet architecture. Both layers trainable parameters are shown in the Figure 4.12



(a) First Layer



(b) Second Layer

Figure 4.12: Parameters of Fully Connected Layers

4.1.4 Activation Layer

In the human brain network, the neuron fires based on the input spike to that neuron. Similarly, an AF is used in an ANN, acting as a decision function. The AF brings non-linearity to the network. There are different types of AF used in the network. The majorly used activation function is given below

1. Hyperbolic Tangent (Tanh): Tanh function has an S-shaped curve shown in Figure 4.13. The output of the function lies between -1 and +1.

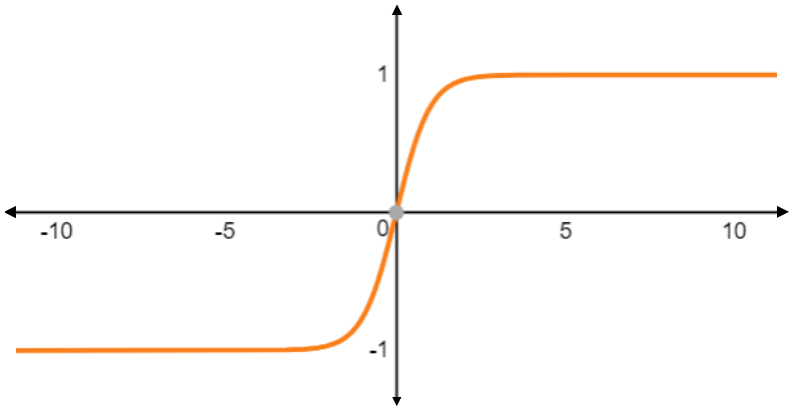


Figure 4.13: Hyperbolic Tangent Activation Function

The mathematical equation is given as

$$f(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{4.7}$$

The advantage of the hyperbolic tangent function is that its output contains positive, negative, and zero values. The mapping becomes easy and makes the function suitable for the classification between two classes. However, it suffers from the vanishing gradient problem. The gradient becomes very small after a particular value of x in which there is a slight change in the y value for x.

2. Sigmoid or Logistic: It has a similar S-shaped curve output shown in Figure 4.14. The output of the sigmoid function lies between 0 and 1.

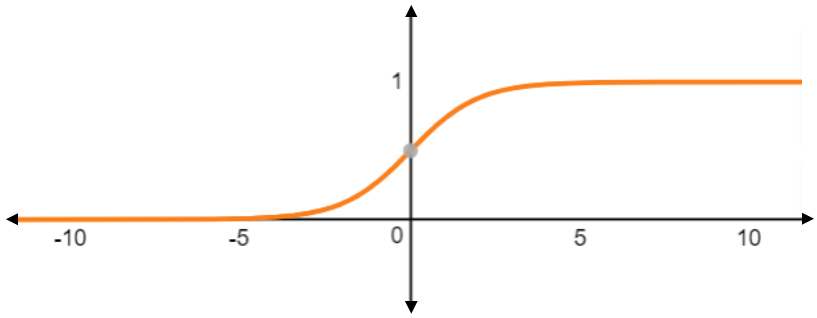


Figure 4.14: Sigmoid Activation Function

The mathematical equation is given as

$$f(x) = \frac{1}{1 + e^{-x}} \tag{4.8}$$

It is widely used when we need to give the output as the probability function. Also, the function is differentiable at all points. However, the sigmoid function also suffers from the vanishing gradient problem.

3. Rectified Linear Unit (ReLU): Currently, this is the most commonly used activation function. The ReLU function is biologically more relevant than the sigmoid and hyperbolic tangent function. The mathematical equation is given as

$$f(x) = max(0, x) \tag{4.9}$$

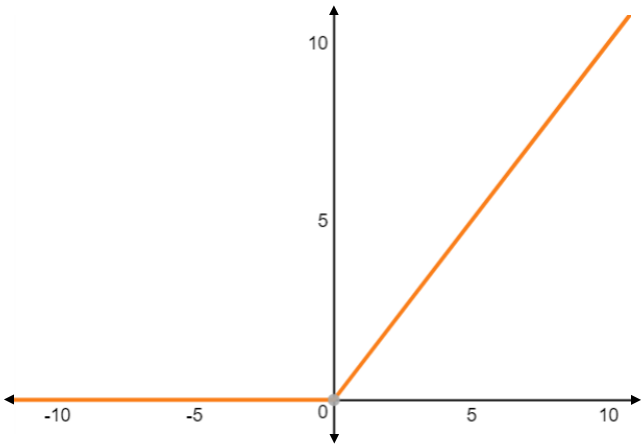


Figure 4.15: Rectified Linear Unit Activation Function

The function passes the positive value and pulls the negative inputs to zero as shown in Figure 4.15. The advantage of the ReLU function is that it is pretty simple to implement, making the network light and increasing the training speed. Also, it removes the problem of vanishing gradient, which occurs in sigmoid and hyperbolic tangent. It is not used in the output layer where we want the probability for each class.

4. Softmax: It is usually used in the output layer of the network. It takes a vector of real values and converts them to the values between zero and one. It converts all the values such that it sums to 1. It is used to give the probability of each class in the output layer. If the input value is small, it converts into a small probability, and if the input value is high, it gets converted into large probability values. The mathematical equation is given as

$$\sigma(\vec{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{L} e^{x_j}} \tag{4.10}$$

where,

$\vec{x}$ - is the input vector to the function

x_i - All x_i values are the elements of the input vector. It can take any real value.

e^{x_i} - The exponential function is applied to all the elements of the input vector. The function will give a small value close to zero when the element value is small and give a large value when the element value is large. It will always give a value above zero. The range of this function is $[0, \infty]$

L - is the number of classes present in the output

$\sum_{j=1}^{L} e^{x_j}$ - is the normalization term. It makes all the output terms in the range (0,1) such that all the values sum to 1.

The Softmax function is also called the multi-class logistic function because softmax is a generalization of the logistic function. It is used for multi-class classification. Softmax can be used for classification only when the classes are mutually exclusive.

4.2 Hardware Implementation of the LeNet architecture

The LeNet architecture is designed on the field-programmable gate array (FPGA) board. The board we used is the ZedBoard Xilinx Zynq®-7000, all programmable system on chip (SoC). The complete details on the tool used and board is given in Chapter 5. The architecture has been designed using the hardware descriptive language (HDL) Verilog. The whole architecture register transfer level (RTL) schematic generated by the HDL in the tool Vivado is shown in the Figure 4.16.

The structure is designed in a hierarchical pattern. For each layer in the LeNet architecture, two modules are designed named as control and execution. The control module generates all the control signals necessary for the operation depending upon the clock and other module operations. The execution module on getting the control signal executes the operation defined in that particular layer. The MAC is the primary unit of the architecture, so a separate module is defined for the MAC operations for all layers. By designing a separate module, it is easy to replace the conventional MAC with the proposed MAC and analyze its impact. Block random access memory (BRAM) is used in the architecture to store the intermediate results of each layer. Distributive read only memory (ROM) is used to store the weights used by each layer.

The naming convention used is as follows, conv is used to represent the convolution layer. The pool is used to represent the pooling layer, fc is used to represent the fully connected layer. The control and execution module of each layer is named ctrl and exec, respectively. The MAC operations performed in the module defined as shared MAC. f stands for feature, which is used to store pictures and intermediate results of each layer; w stands for weight, which is used to store the weights used by each layer. The number following f/w represents the number of layers where RAM or ROM is located. The network has a total of 7 layers. For example, f1 ram represents the storage location of the input picture of the first layer. The weights are only used in the convolution layers and the fully connected layers, so there are w1_rom, w3_rom,

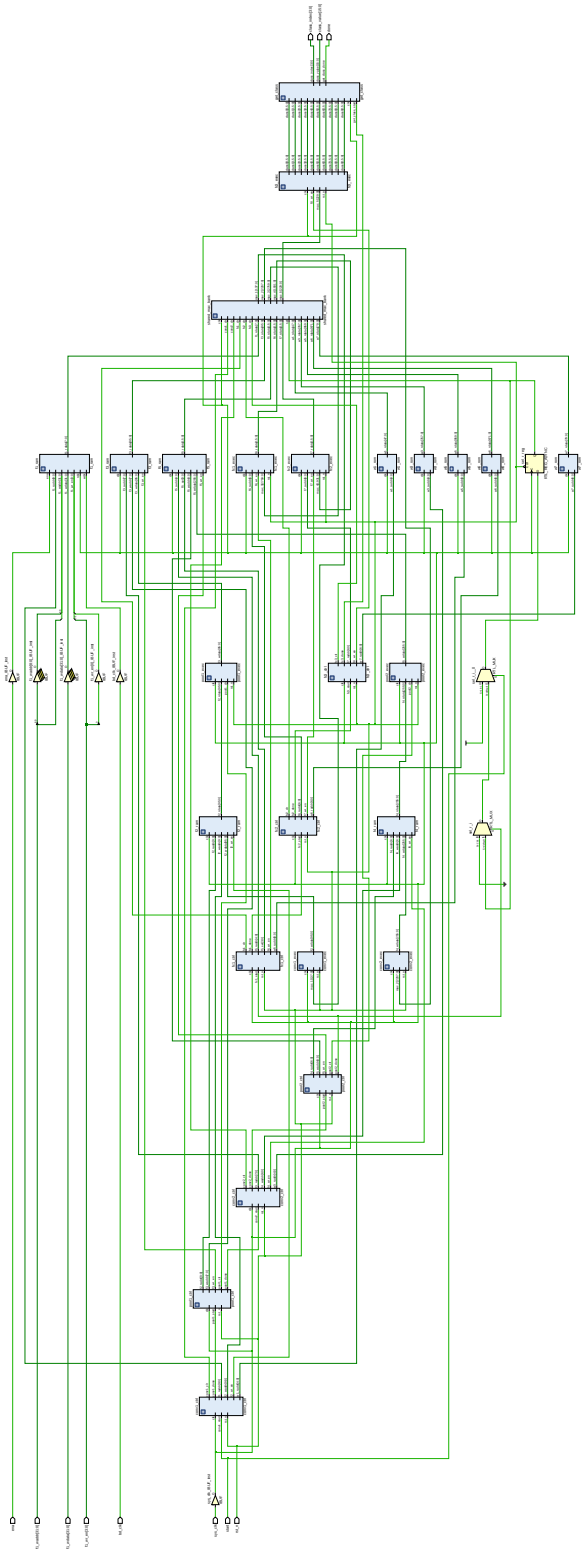


Figure 4.16: RTL Schematic of LeNet architecture

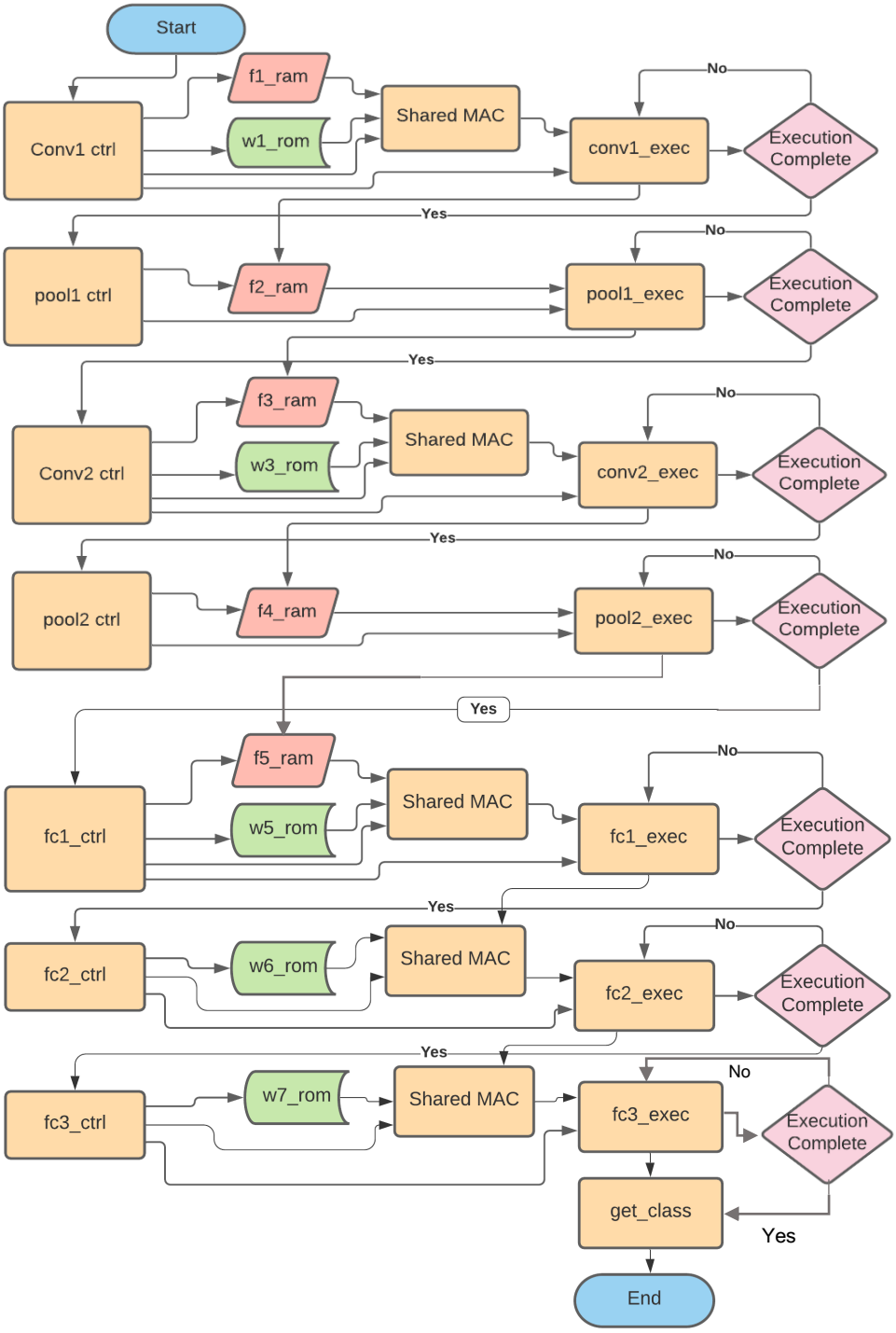


Figure 4.17: Flowchart for Hardware Implementation of LeNet Architecture

w5_rom, w6_rom, and w7_rom. The final module is named as get class which will give the final output of the architecture.

The complete data flow of the designed architecture is explained in the flowchart shown in Figure 4.17. The first module is conv1 ctrl which represents the first convolution layer control block. The control logic scans line by line according to the time sequence and gives the corresponding input feature read address and weight read address, and the write address of the output buffer. The feature read address is given to the f1_ram module, where the input data is stored. The weight read address is given to the w1_rom, containing the weight for the first layer convolution. The input feature and weight data are given as input to the shared mac module, which performs the MAC operation. The conv1_exec module represents the execution module for the first convolution layer. The execution block performs the ReLU operation and writes the output feature data to the f2_ram. When the execution of the first layer is completed, the flow moves to the first pooling layer.

The first pooling layer control module (pool1_ctrl) generates the read and write addresses of the memory to scan the entire input feature map. The input feature is available in the f2_ram, which obtain after the first convolution layer. The calculation is done in the execution block of the first pooling layer(pool1_exec). We have used the max pooling operation in the pooling layer. In the max-pooling operation, the computing unit is the comparator. When the execution of the first pooling layer is completed, the output feature is stored in the f3_ram, and the flow moves to the second convolution layer.

The second convolution layer data flow is the same as the first convolution layer. The weight for the second convolution is stored in the w3_rom. After completion of the second convolution layer operation, the flow moves toward the second pooling layer. The second pooling layer executes similarly as the first pooling layer. The output feature from the second pooling layer is stored in the f5_ram block. After the second pooling layer, the flow moves towards the fully connected layers.

The fc1 represents the first fully connected layer which has 120 nodes in the LeNet architecture. The fully connected layer is very similar to the convolution layer. The

difference is that the number of weights is comparatively more in fully connected layers. In addition, the output result of the fully connected layer needs to be used at the same time in the next layer, so it is not stored in the RAM and directly stored in the register. The fc2 represents the second fully connected layer, and fc3 represents the third fully connected layer which is also the output layer of the architecture. The fc3 layer has ten output nodes. Since the fc3 layer is the last layer of the network, the get_class module compares the maximum value of the output and gives the final classification result.

Chapter 5

Experiment Evaluation

In this chapter, we have elaborated on the design approaches and tools used to design, verify and implement the architecture mentioned in the previous chapters. There were two design approaches we have investigated.

First, for the MAC unit, a semi-custom VLSI circuit design approach is used. In this approach, the different bit precision MAC unit is designed with and without power gating. Then the physical parameters of the proposed MAC are compared with the state-of-the-art MAC for the analysis. The second approach is for the complete LeNet architecture using the proposed MAC. The LeNet architecture is designed, verified, and implemented on the FPGA board.

5.1 Semi-Custom ASIC approach for MAC unit

For the MAC design and analysis, we have followed a semi-custom ASIC approach. The design flow and the tools used are explained in the following subsections.

5.1.1 Design Flow

The design flow of the semi-custom approach is shown in Figure 5.1. The first step towards design is to define the design specification. Then according to the design specification, the proposed MAC architecture is presented in Verilog hardware descrip-

tion language(*HDL*). The RTL schematic is generated using the HDL code. Then the functionality of the design is verified using simulation in Xilinx *Vivado* tool.

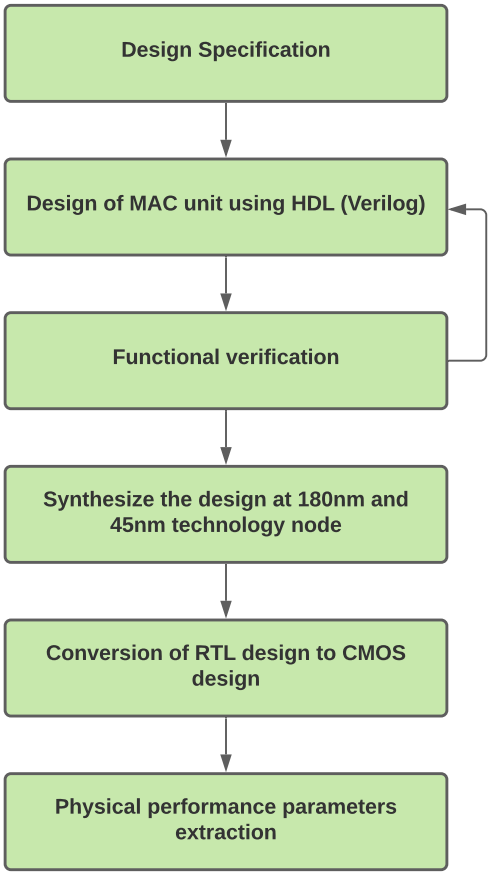


Figure 5.1: Design Flow for semi-custom circuit design approach

Addressing ASIC design, the RTL for our proposed 8-bit precision MAC architecture is synthesized, and results obtained by *Design Compiler*-Synopsys [51]. The compiler's netlist file is educed using *Encounter*-Cadence. The .cdl file generated then use for the conversion of RTL digital design into CMOS design using the *v2lvs*-Mentor Graphics [52]. The CMOS extracted design is simulated in *Cadence-Virtuoso* [53] and extracted performance parameters at typical typical (TT) process corners and mismatch. Addressing to the technology scaling impact, the multiply-accumulate architecture is synthesized at technology nodes of *45nm* and *180nm*. Furthermore, to see the impact of static power dissipation, two experiments are performed i.e. with and without PG. The physical performance parameters are extracted for the proposed

MAC architecture with and without the power-gating technique and for the related work.

5.1.2 Tools

The brief about the tools used in the semi-custom approach is in the following subsections.

5.1.2.1 Xilinx Vivado

The first tool which is used is the Vivado Design Suite by Xilinx [54]. The tool Vivado has an integrated design environment (IDE). It provides various tools from the system level to the integrated circuit (IC) level. The key features of the tool are to simulate, synthesize and implement the HDL design.

We have used the Vivado 2019.1 version for our research. In the semi-custom approach, we have designed our MAC on the Vivado tool. After designing, we have generated the RTL of the design. We used the simulation feature of the tool to verify the functionality of the design.

5.1.2.2 Synopsys Design Compiler

Synopsys provides various tools for the synthesis process. The design compiler is the core product of the synthesis. It optimizes the design and provides an efficient representation of the design in terms of physical parameters. It optimizes and synthesizes both combinational and sequential devices. The design flow of the compiler is illustrated in Figure 5.2.

The design compiler takes the HDL file and converts that into a technology-dependent optimized gate-level netlist. The task involves in the flow of the design compiler synthesis is

1. Reading the source file: Two files are given as an input to the compiler. First is the technology library file, which consists of the complete characterization of all standard cells for the defined technology node. The second is the HDL file

which contains the design. The HDL file is read in two phases named analysis and elaboration. In analysis, the compiler reads the file and looks for the syntactical errors. After analysis in elaboration, the design is translated into a technology-dependent design.

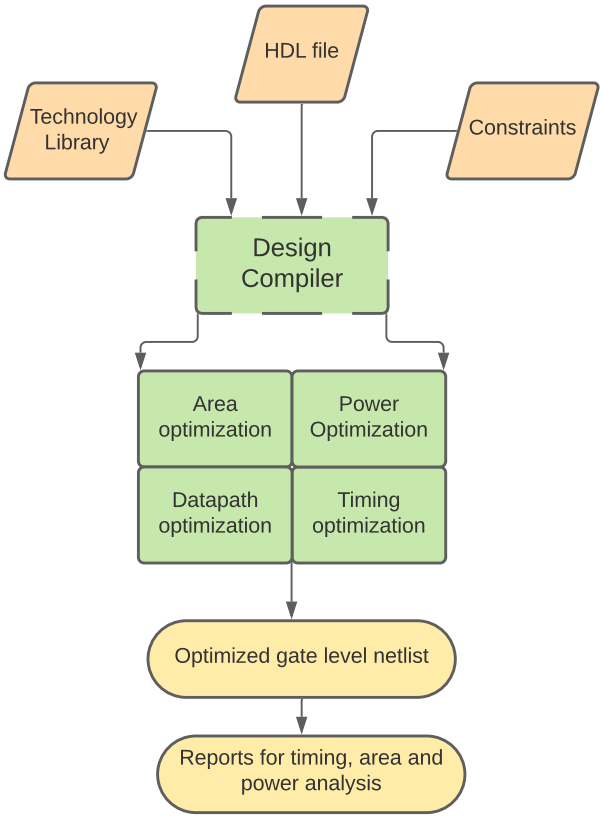


Figure 5.2: Design Flow of Synopsys design compiler

2. Applying the constraints: In this, the constraints of the design are given. The constraints mainly apply to the central clock of the design. Also, to simulate a natural behavior, some uncertainty parameters are given.

3. Optimizing the design: The compiler looks for all the possible optimization of the physical parameters and finally generates an optimized gate-level netlist.

4. Generating reports: At last, timing, power, and area analysis of the optimized design reports were generated. These reports consist of vital physical parameters

like area utilization, static and dynamic power consumption, and critical path delay in the design.

For our research, we have used two technology library files, the first one at *180nm* technology node and the second one at *45nm* technology node.

5.1.2.3 Mentor graphics- V2LVS

The v2lvs is the acronym for Verilog to LVS (Layout vs. Schematic). Mentor Graphics provides the v2lvs tool. It is used to translate the Verilog structured file into a spice-level netlist file. The flow of the tool is illustrated in Figure 5.3.

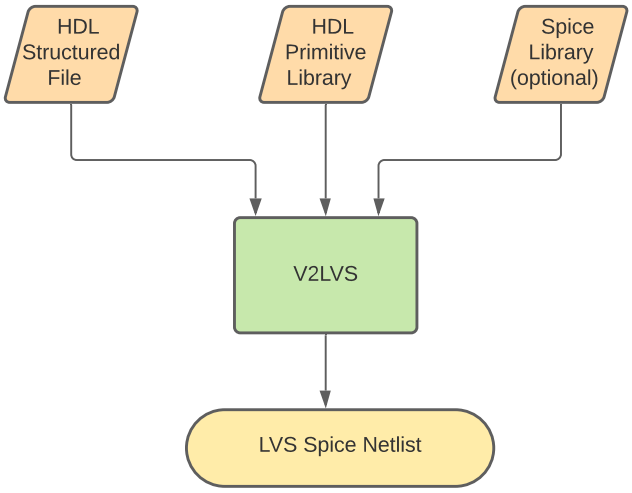


Figure 5.3: Design Flow of verilog to LVS

The tool can have three inputs, out of which two are mandatory, and one is optional. The first mandatory input is HDL structured file. In our case, it is a Verilog design file that contains the module information. The second input is the Verilog primitive library file. It gives the information of pins associated with nets shown in positional cell instances. The third input is the spice library which is an optional input. This tool gives an LVS spice netlist which can be used as an input in Calibre LVS.

5.1.2.4 Cadence- Encounter and Virtuoso

The Cadence tools work on the complementary metal-oxide-semiconductor (CMOS) level schematic, layout, verification, and simulation. We have used the Cadence tools for the implication of the power gating technique in the MAC unit. The spice level netlist from the v2lvs is imported in the Cadence- Virtuoso to work on the CMOS level design.

5.2 FPGA based approach for LeNet architecture

The MAC unit we have analyzed through a semi-custom digital design approach. For the implementation of the LeNet architecture, we have used the FPGA-based approach. The design flow for the FPGA-based approach and the tools used are explained in the following subsections.

5.2.1 Design Flow

The design flow for the FPGA-based approach is illustrated in Figure 5.4. First, the design specification is decided. Based on the design specification, the design is described using the HDL. We have used the Verilog HDL for designing. The top-down hierarchy is used for the LeNet architecture. Then RTL for the design is generated using an HDL file. The RTL schematic of the design using HDL is explained in Section4.2. Before moving to the RTL synthesis, the design is verified that functionally it is working or not. The verification carries out using the functional simulation. If there is an error in the design functionality, then the flow moves back to the design description, and the design is rewritten to mitigate the errors. Once the design is functionally correct, the RTL synthesis is performed. In synthesis, the RTL level designed is converted into gate-level representation. The tool Vivado optimizes the synthesis process for performance and memory usage. After synthesizing, the design is again checked for functionality. Also, a timing analysis can be performed. If there is any timing violation or functionality error, the design is again described, and the same

process follows till synthesis till it is functionally correct with no timing violation.

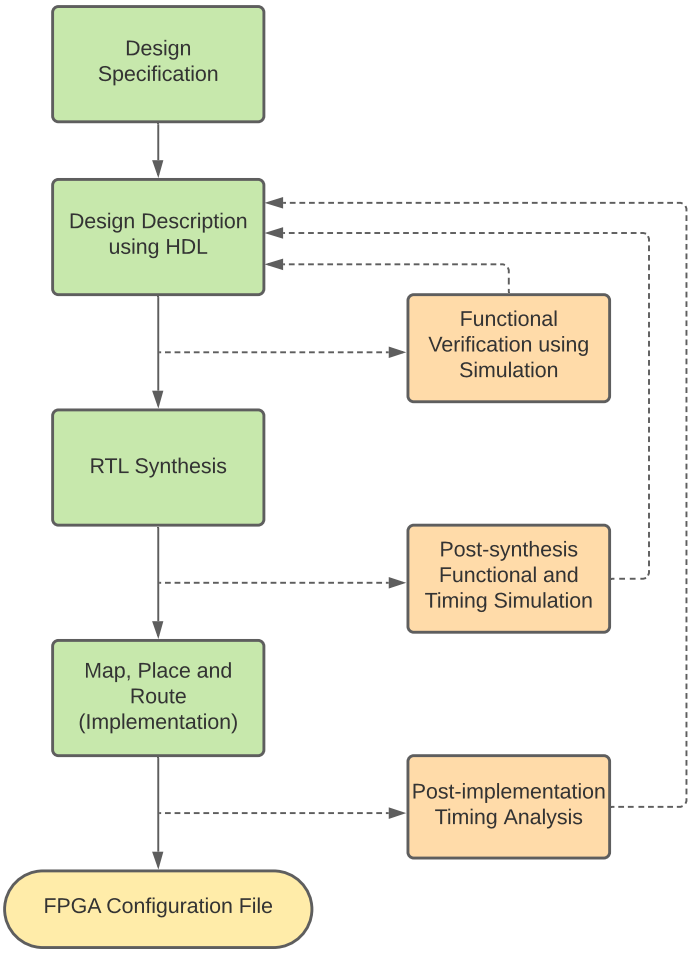


Figure 5.4: Design Flow for FPGA based approach

The last step is the implementation of the synthesized design. In this, the netlist is placed and routed on the FPGA device's resources. During implementation, the design's logical, physical, and timing constraints are considered in mapping. After implementation, post timing analysis can be performed to check if there is any timing violation present. After successful implementation, the FPGA configuration file is generated, which is used to transfer the design into the FPGA board.

5.2.2 Tools

The Xilinx Vivado tool is used for the complete FPGA-based approach. Vivado provides an integrated environment where each step of the flow explained in section 5.2.1 can be executed. The Vivado tool is used to design the LeNet architecture, synthesize it, and implement it on the FPGA board. In the following subsection, the details on the FPGA board used and the IP used for the design are explained.

5.2.2.1 FPGA board

For our research, we have used Zedboard xc7z020clg484-1 FPGA [55]. ZedBoard is a low-cost development board for the Xilinx Zynq-7000 SoC. The specification of the FPGA board is shown in Table 5.1.

Table 5.1: Specification for the Zedboard - XC7Z020clg484-1 FPGA

Specification	**Count**
LUT elements	53200
LUT RAM	17400
Flip Flops	106400
I/O Block	200
I/O pin count	484
Block RAM	140
DSP	220
Reference Operating Voltage	0.95 V

5.2.2.2 IP used in the design

For designing the LeNet architecture, we have used four existing IPs in the Vivado. The IPs are the features that the Vivado tool provides. We can directly use the existing IP and get the functionality in the design. IP is designed with the optimized method. The four IPs which are used in our LeNet architecture design are as follows

1. Block Memory Generator: The block RAM(BRAM) is the onboard memory with variable width and height. The block memory generator is the IP that generates the memory using the embedded block memories primitives [56]. The memories which are generated are performance and area optimized memories. With a block memory generator, five types of memory can be generated.

   - Single-port RAM
   - Simple dual-port RAM
   - True dual-port RAM
   - Single-port ROM
   - Dual-port ROM

   We have used simple port dual RAM to store the input and the intermediate layer results. Simple dual-port RAM is used to utilize the minimum resource of FPGA. The modules f1_ram, f2_ram, f3_ram, f4_ram, and f5_ram are BRAM we have used in the LeNet architecture design.

2. Distributed Memory Generator: The distributed memory generator generates the memories using LUT RAM resources [57]. The generator offers a variety of memories. We have used the distributive ROM for storing the weight values for the convolution and fully connected layers. The modules w1_rom, w3_rom, w5_rom, w6_rom, and w7_rom used in the design of LeNet architecture are distributive ROM IP.

3. Multiplier: The multiplier IP is used to implement a resource efficient high performance multiplier [58]. This IP is used in all applications where fixed-point or integer multiplication is required. The multiplier IP is used in the shared MAC module in the LeNet architecture. The schematic symbol of the multiplier IP is shown in Figure 5.5. The port description is as follows:

   A - It is the input bus with dimension [N-1:0],

   B - It is the input bus with dimension [M-1:0],

CLK - It is the rising edge clock input,

CE - It is active high clock enable input,

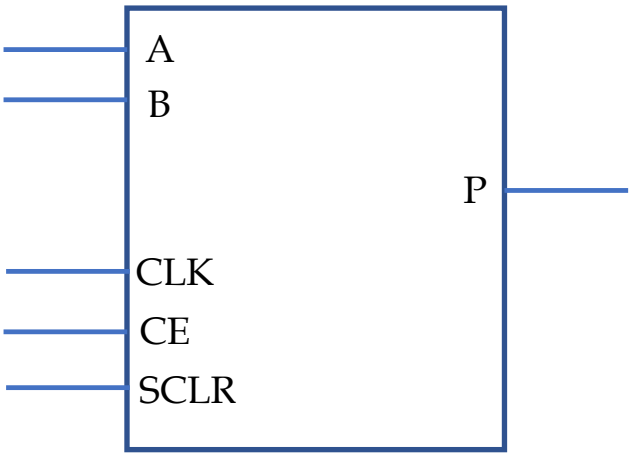


Figure 5.5: Multiplier IP Schematic Symbol

SCLR - It is active high synchronous clear input,

P - It is the product output

4. Adder/Subtracter: The adder/subtracter IP is used to perform the addition or subtraction operations. It implements high-performance and area-efficient adders and subtracters [59]. The IP can be customized to use either DSP slice or FPGA logic for implementation. The schematic symbol of the adder/subtracter IP is shown in Figure 5.6. The port description is as follows:

   A - It is the input bus with dimension [N:0],

   B - It is the input bus with dimension [M:0],

   C_IN - It is the input carry,

   ADD - It is a control signal which defines which operation will be performed addition or subtraction,

   BYPASS - It loads the B port onto the S port,

   CLK - It is the rising edge clock input,

   CE - It is an active-high clock enable input,

SCLR - It is an active-high synchronous clear input

S - It is the output bus which is the sum in case of addition

C_OUT - It is the output carry

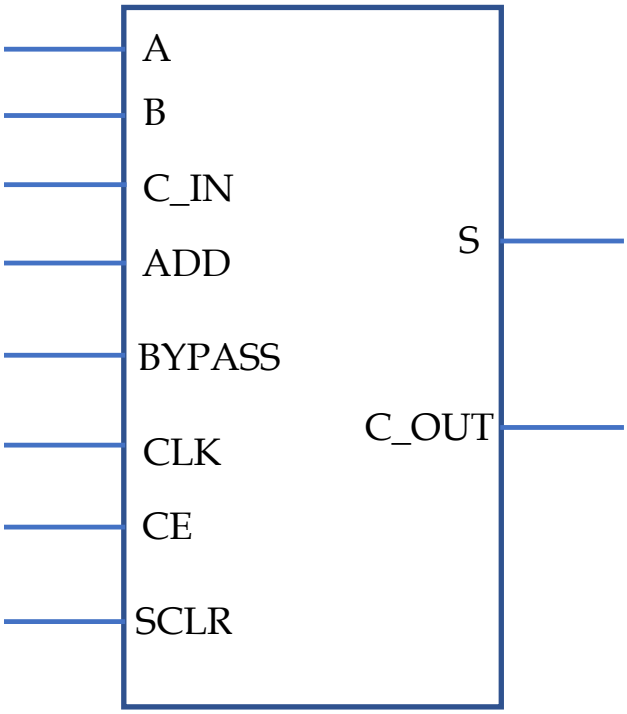


Figure 5.6: Adder/Subtracter IP Schematic Symbol

Chapter 6

Results and Discussion

The objective of the work is to design an efficient MAC unit. The MAC unit is the dominant part of the neural network, taking most of the power and chip area. We designed the MAC unit with efficient performance in terms of physical parameters without compromising the accuracy. We observed the impact of technology scaling on circuit delay and saturation current. For proposed logic, results from extraction and observation are made for both single bitcell processing and an array of bitcell. With the help of the efficient MAC, the neural network performance can be improved. We have implemented the LeNet architecture on the FPGA board to analyze the network's performance with the efficient MAC unit. The MAC unit and LeNet architecture analysis and performance results are discussed in the following sections.

6.1 Bit-precision Computation and accuracy impact using benchmark LeNet and CaffeNet

The training of the DNN architecture is performed first on the LeNet benchmark for MNIST and CIFAR-10 datasets and then on the CaffeNet benchmark for the ImageNet dataset. The training is executed for different bit precision starting from 32-bit and then reducing by a factor of two till 2-bit. Table 6.1 shows the training accuracy for three data sets at different fixed point bit precision. ImageNet dataset

consists around 14 million images whereas MNIST dataset consist of around 60,000 images. As the CaffeNet architecture is used for the ImageNet dataset the training accuracy for CaffeNet is lower than the LeNet architecture. There is a shallow (1%) loss of accuracy between usage of 8-bit and 32-bit fixed points. Sacrificing 1% of accuracy will allow for 4× overall memory bandwidth reduction. Based on the observation, 8-bit precision architecture is selected and compared with the state-of-the-art.

Table 6.1: Comparison of training accuracy for different bit-precision used in different DNN architecture

Fixed Point	Training Accuracy (%)		
Data Precision	LeNet		CaffeNet
Representation	MNIST	CIFAR-10	ImageNet
32-bit	99.1	81.7	56.9
16-bit	98.7	81.2	56.8
8-bit	98.2	80.7	56.7
4-bit	97.6	79.6	06.0
2-bit	85.9	48.0	00.1

6.2 Bit-serial computation based 8-bit precision MAC unit performance

The proposed 8-bit MAC architecture is compared with the various state-of-the-art architectures on the important performance parameters like area, static and dynamic power, and critical path delay. The physical parameters for the proposed 8-bit MAC unit and state-of-the-art at higher technology node i.e *180nm* and lower technology node i.e *45nm* are shown in Table 6.2 and Table 6.3 respectively. The area and delay product comprises a performance metric known as area-delay-product (ADP)

Table 6.2: Performance parameter metrics for 8-bit precision proposed Bit-cell based MAC architecture and state-of-the-art at 180nm technology node.

Performance Parameters	**IEEE Lib. [35]**	**Array Mult. [25]**	**Wallace Tree [27]**	**Shift-Add [31]**	**Vedic Math [39]**	**Proposed MAC without PG**	**Proposed MAC with PG**
Area (μm^2)	4948.28	4932.50	5067.31	9150.93	6105.53	3631.26	3813.52
Dy. Power (mW)	1.98	1.89	2.10	0.74	2.14	1.97	2.02
St. Power (nW)	49.36	50.56	51.71	115.09	64.72	38.05	31.4
Critical Delay (ns)	20.76	26.98	21.00	15.60	25.05	16.94	17.06
ADP(fm^2s))	102.70	133.00	106.40	967.90	152.90	61.51	65.05
Integer	signed	signed	signed	unsigned	signed	signed/unsigned	signed/unsigned

for comparing the proposed architecture with state-of-the-art.

At *180nm* among all of the state-of-the-art architectures, IEEE Library architecture [35] has the lowest ADP. We also observed that our bitcell based architecture without PG technique ADP is 40.10% less than the IEEE library. With power gating, there is a 5.75% increase in ADP compared with the without PG architecture, but it still has 36.67% less ADP than IEEE standard library architecture. The static power of the IEEE Library architecture is the lowest among the state-of-the-art architectures. We observed that the bitcell based MAC without PG static power is 22.91% less than the IEEE library architecture. The power gating technique main advantage is on the static power parameter. We observed that with PG, there is even 17.47% more decrement in the static power than the without PG architecture. Compared with the IEEE library architecture, there is a total save of 36.38% in the static power with PG architecture. The dynamic power of bitcell based MAC architecture is comparable with the state-of-the-art architecture.

At *45nm*, the Wallace tree has the least ADP among all state-of-the-art architectures as shown in Table 6.3. The proposed circuit without the PG technique has 34.35% less ADP than Wallace tree architecture. In the PG technique, there is an increment of 21.30% in APD compared to without PG technique architecture. The PG technique's proposed circuit has 20.36% less ADP than the Wallace tree 8-bit MAC architecture. The IEEE library architecture has the least static power among

Table 6.3: Performance parameter metrics for 8-bit precision proposed Bit-cell based MAC architecture and state-of-the-art at 45nm technology node.

Performance Parameters	**IEEE Lib. [35]**	**Array Mult. [25]**	**Wallace Tree [27]**	**Shift-Add [31]**	**Vedic Math [39]**	**Proposed MAC without PG**	**Proposed MAC with PG**
Area (μm^2)	832.06	862.57	838.16	890.73	1249.26	616.19	723.58
Dy. Power (μW)	495.52	475.58	497.93	200.6	527.47	529.52	542.31
St. Power (μW)	3.81	3.98	3.88	5.48	5.76	2.49	1.80
Critical Delay (ns)	6.71	7.43	6.44	10.98	7.25	5.75	5.94
ADP(fm^2s))	5.583	6.408	5.397	9.78	9.06	3.54	4.30
Integer	signed	signed	signed	unsigned	signed	signed/unsigned	signed/unsigned

state-of-the-art architectures at *45nm*. We observed that the bitcell based MAC without PG static power is 34.64% less than the IEEE library architecture. The bitcell based architecture with PG has 27.71% less static power compared with without PG architecture. On comparing, the bitcell based architecture with PG has 52.75% less static power than the IEEE library architecture. The bitcell based architecture with power gating has shown more improvement with the scaling of the technology node.

We have used the column-based coarse-grain power gating technique. In the coarse-grain power gating technique, the proposed 8-bit MAC required the number of MOS's is the same as used in the single bitcell with a change in MOS channel width due to the driving load. The advantage of using the coarse-grain PG technique is that the relative impact of area increase in 8-bit MAC is much smaller when compared with the single power gated bitcell. For a single bitcell, there is a 147.5% & 319.0% increase in area with PG technique, while for 8-bit MAC, there is just 5.01% & 17.42% increase in area at *180nm* and *45nm* respectively. Overall it shows that the proposed architecture has better performance than state-of-the-art, and it can be implemented using a semi-custom approach for chip system design.

6.3 Higher bit precision-based multiply-accumulate unit performance and comparison

The proposed bitcell based MAC has been named as *BitMAC*. The 8-bit precision *BitMAC* shows promising performance compared with the state-of-the-art architectures. Furthermore, we have explored the performance of the *BitMAC* for higher bit precision to validate the physical parameter impact compare to the state-of-the-art. The *BitMAC* is designed for 8, 12, and 16-bit precision. The *BitMAC* is compared with the standard IEEE Library-based MAC unit design architecture [35]. The comparison for the physical performance parameters such as area, delay, dynamic power, and static power for proposed *BitMAC* and IEEE library-based MAC unit at *180nm*, and *45nm* technology node is shown in the following subsections. It is observed that the proposed *BitMAC* is the desirable choice at lower as well as higher precision multiply-accumulate computation in hardware implementation deep neural-network accelerator.

6.3.1 Logic area utilization

The area utilization comparison of MAC unit using proposed method and IEEE library architecture [35] for 8-bit and higher bit precision computation at *180nm* and *45nm* technology nodes is shown in Figure 6.1.

At *180nm* on increasing the precision from 8-bit to 12-bit there is an increase of 5961 (μm^2) in IEEE architecture while in *BitMAC* 4816 (μm^2) has increased. Also, changing the precision from 12-bit to 16-bit, there is an increase of 7627 (μm^2) in IEEE architecture while in *BitMAC* 6536 (μm^2) is increased.

At *45nm* on increasing the precision from 8-bit to 12-bit there is an increase of 949 (μm^2) in IEEE architecture while in *BitMAC* 755 (μm^2) has increased. Also, changing the precision from 12-bit to 16-bit, there is an increase of 2056 (μm^2) in IEEE architecture while in *BitMAC* 994 (μm^2) is increased.

It is observed from the figure that on increasing the bit precision, the area utilized

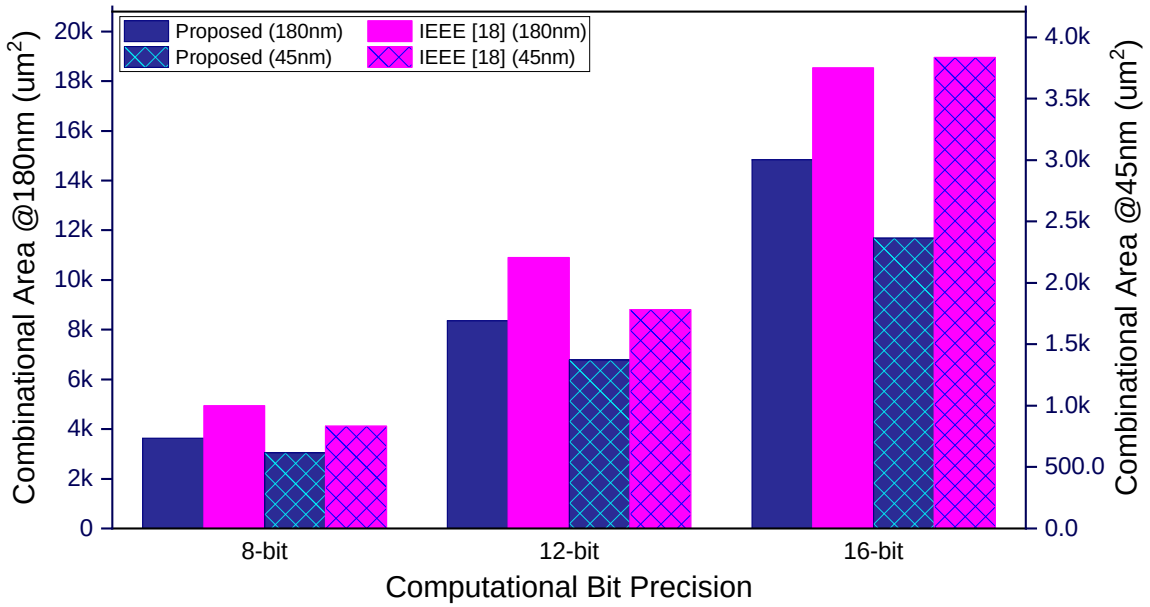


Figure 6.1: Combinational logic area utilization for different bit-precision computation at *180nm* and *45nm* technology node

by the *BitMAC* is always less than the state-of-the-art architecture for both technology nodes. Thus it is fair enough to say that *BitMAC* is the most efficient MAC in terms of area utilization.

6.3.2 Logic critical path delay

The critical path delay comparison of MCA unit using proposed method and IEEE library architecture [35] for 8-bit and higher bit precision computation at *180nm* and *45nm* technology nodes is shown in Figure 6.2.

At *180nm* on increasing the precision from 8-bit to 12-bit, there is a 32.80% increase in IEEE architecture delay while in *BitMAC* there is 28.39% increment in delay. Also, changing the precision from 12-bit to 16-bit, there is a 34.02% increase in IEEE architecture delay while in *BitMAC* the delay increases 14.80%.

At *45nm* on increasing the precision from 8-bit to 12-bit, there is a 20.26% increase in IEEE architecture delay while in *BitMAC* there is 16.17% increment in delay. Also, changing the precision from 12-bit to 16-bit, there is a 19.95% increase in IEEE architecture delay while in *BitMAC* the delay increases 10.02%.

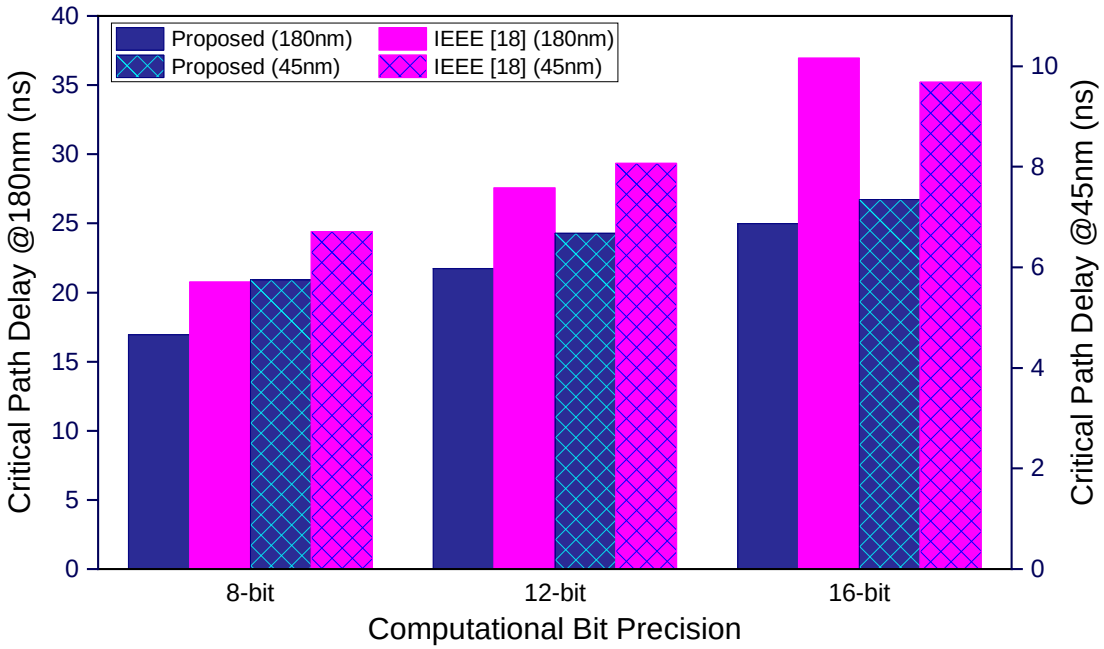


Figure 6.2: Logic circuit critical path delay for different bit-precision computation at *180nm* and *45nm* technology node

It is observed from the figure that on increasing the bit precision, the critical path delay for the *BitMAC* is always less than the state-of-the-art architecture for both technology nodes.

6.3.3 Dynamic power consumption

The dynamic power consumption of MAC unit using proposed method and IEEE library architecture [35] for 8-bit and higher bit precision computation at *180nm* and *45nm* technology nodes is shown in Figure 6.3.

It is observed that the dynamic power of *BitMAC* for 8-bit and 12-bit is comparable to the IEEE library architecture. At 16-bit, the dynamic power is for *BitMAC* is slightly lower than IEEE library architecture. Overall in *BitMAC* dynamic power, there is neither positive nor negative effect compared with the state-of-the-art architecture.

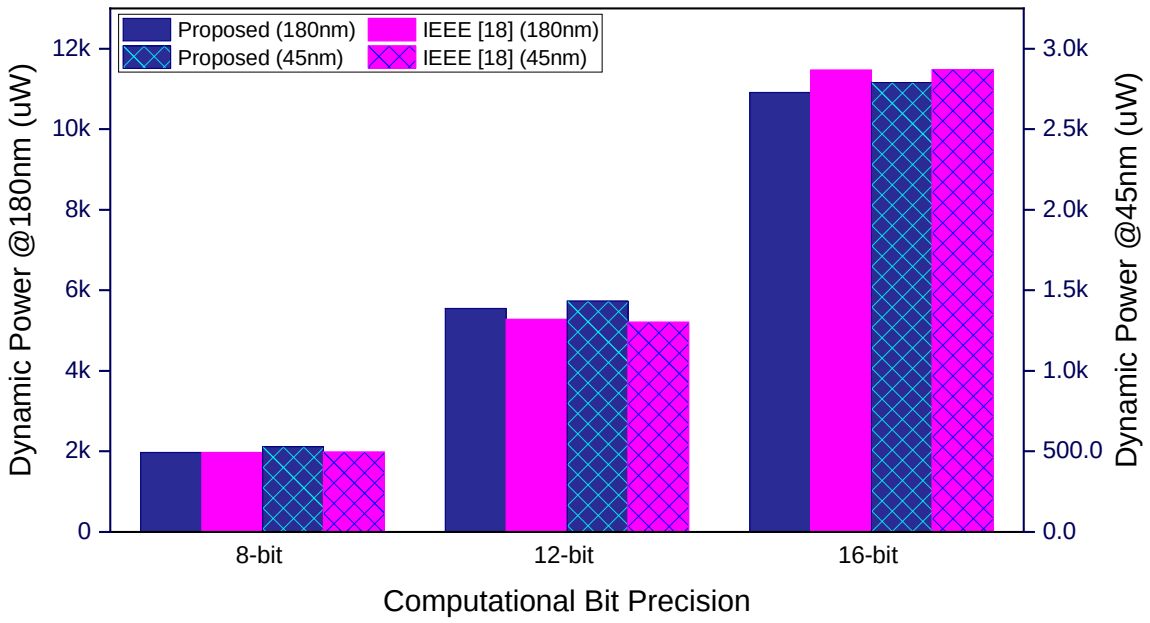


Figure 6.3: Total dynamic power consumption for different bit-precision computation at *180nm* and *45nm* technology node

6.3.4 Static power consumption

The static power consumption of MAC unit using proposed method and IEEE library architecture [35] for 8-bit and higher bit precision computation at *180nm* and *45nm* technology nodes is shown in Figure 6.4.

It is observed that *BitMAC* static power for all the bit precision is lower than the state-of-the-art power dissipation. At *180nm* on average *BitMAC* static power is 18% less than IEEE library architecture. With technology scaling, this improvement number increases. At *45nm* on average *BitMAC* static power is 35% less than the state-of-the-art. Thus, the *BitMAC* architecture is performance efficient on higher as well as lower technology nodes.

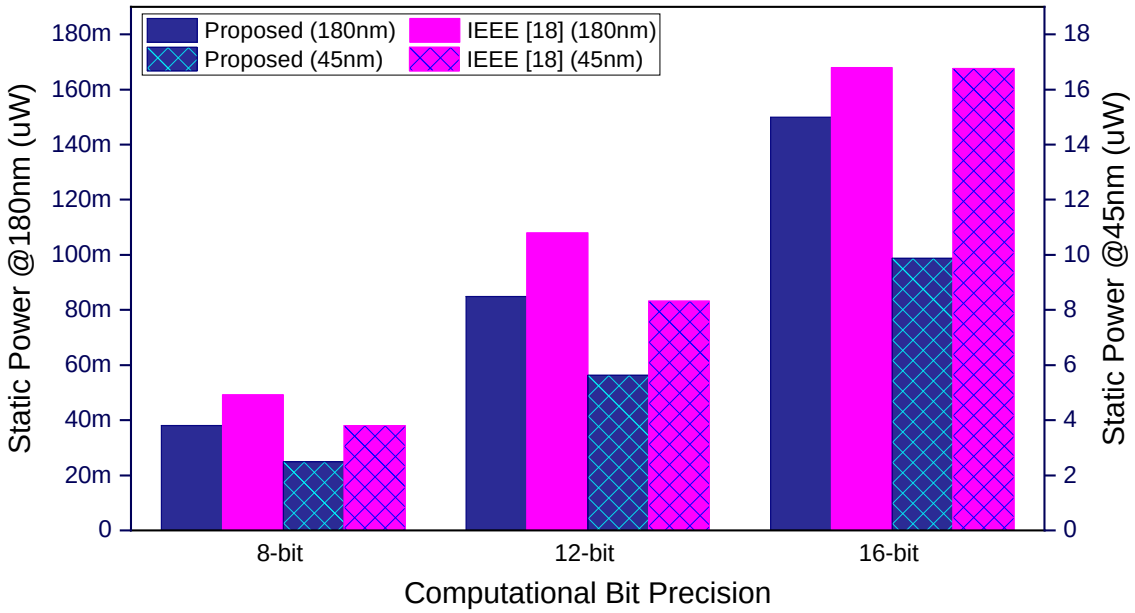


Figure 6.4: Total static power consumption for different bit-precision computation at *180nm* and *45nm* technology node

6.4 Performance of the Lenet architecture with proposed MAC unit

The LeNet architecture is implemented on the FPGA board using the FPGA-based design approach. The two architectures are implemented. The first one uses the conventional IEEE MAC unit, and the second one uses our proposed bitcell based MAC unit ($BitMAC$). The comparison of the resource utilized by both the designs is given in the Table 6.4. Out of all the resources listed in the table, the critical resource is the look-up table (LUT). In the FPGA board, the computation is performed using the LUT approach. The LUT is one parameter that needs to be less used, to design a resource-efficient architecture. It is observed that with $BitMAC$, we were able to reduce LUT utilization up to 20% in the Zedboard. The flip-flops usage is increased by 5.26% with $BitMAC$. The flip-flops availability on the board is much higher than the LUT. With the increase of 5% flip-flops usage, there is no significant effect on the design utilization parameters. The MAC unit does not affect the other resources listed

in the table.

Table 6.4: Comparison of resource utilization in LeNet architecture with conventional MAC and BitMAC

Resource	LeNet with conventional MAC	LeNet with BitMAC	Avaliable Resource	Utilization % with conventional MAC	Utilization % with BitMAC
LUT	34,914	24,526	53,200	65.68	46.10
LUTRAM	232	228	17,400	1.34	1.30
Flip Flop	18,499	24,105	1,06,400	17.39	22.65
BRAM	33.50	33.50	140	23.93	23.93
IO	68	68	200	34.00	34.00
BUFG	1	1	32	3.13	3.13

As explained in Section 4.2, the shared MAC module is used for all the MAC operations in the design. There are separate MAC units for the fully connected layers in that module, and the convolution layers have shared the MAC unit. The impact of $BitMAC$ and conventional MAC unit on the LUT utilization is shown in Table 6.5. Out of the total 34,914 LUT's utilized by the complete conventional design, the MAC unit alone has taken 26,640 LUT's. In a conventional LeNet design, 76.30% LUT is utilized by the MAC unit only. The total LUT utilized by MAC is divided into three fully connected layers. It is observed that in each fully connected layer there is approximate 50% LUT is saved using the BitMAC.

For timing analysis of the design, we have used a clock period of 15ns. The analysis of the critical timing parameters in both designs is shown in Table 6.6.

We have observed three critical parameters of the design critical path delay, setup slack and hold slack. The critical path delay gives the information of the worst path delay in the overall design. The slack is the difference in the required time and the arrival time of the data. The timing slack gives the information that the design is working without timing violation or not.

The setup slack and Hold slack is given as

$$Setup\ slack = Required\ Time - Arrival\ Time \tag{6.1}$$

Table 6.5: Comparison of look-up-table utilization by MAC unit for fully connected layers in LeNet architecture

Layers	LeNet with conventional MAC	LeNet with BitMAC	% Save
Fully connected layer 1	15,360	7,560	50.78
Fully connected layer 2	10,080	5,292	47.50
Fully connected layer 3	1,200	630	47.50
Total	26640	13482	49.39

Table 6.6: Timing Analysis for LeNet architecture with conventional MAC and Bit-MAC

Parameter	LeNet with conventional MAC	LeNet with BitMAC	Improvement %
Critical delay(ns)	13.77	11.31	17.86
Setup Slack(ns)	0.50	3.09	83.81
Hold Slack(ns)	0.019	0.019	0.00

$$Hold\ slack = Arrival\ Time - Required\ Time \tag{6.2}$$

The required time is when the data has to be present at a flip flop or node. It can also be defined as the time taken by the clock to traverse through the clock path. The arrival time is the actual time at which the data has arrives at that flip fop or node. It can be defined as the time required by the data to traverse through the data path. If the setup slack is negative, there is a setup timing violation, and if the hold slack is negative, then there is a hold timing violation.

The LeNet architecture with $BitMAC$ has shown improvement in the critical path delay and setup slack. There is no improvement on the hold slack with the use of $BitMAC$ in the design.

Chapter 7

Conclusion

This thesis presents a semi-custom ASIC design approach for VLSI design architecture that can implement logic using digital design technique and CMOS design technique that can be better in terms of area, delay, and power. We designed single-bit multiplier architecture called bitcell and used that bitcell for higher precision MAC unit implementation. The semi-custom ASIC design-based proposed model is synthesized at both *180nm* and *45nm* technology node. To analyze the efficiency of the proposed architecture, various performance parameters were calculated and compared with the state-of-the-art. The results show that the proposed architecture is the best choice among all architectures. Further, to address the lower technology node, the coarse-grain power gating technique is used in the architecture to save the static power dissipation. This MAC architecture can be used in all DNN applications, especially where area and power are on a tight budget.

It is noticed that that the accuracy of the DNN increase with increasing the computation elements and number for deep layers. Moreover, using a conventional MAC-based approach deeper neural network is difficult to implement with minimum area utilization. The ADP for different MAC designs is calculated, and it is observed that the proposed approach has a better ADP than the state-of-the-art. Further, the LeNet architecture is design using FPGA based design approach, and it is observed that by using the proposed bitcell based MAC, the resource utilization and the timing analysis is improved compared to the state-of-the-art. The proposed bit serial computing

technique based MAC design will be best suited in the AI-enable applications where the area and power are in the tight budget.

7.1 Future scope of work

We have introduced a bit-serial computing technique based MAC unit which mitigate various physical parameters of the deep neural network. However there are some future coarse of actions that can further improve the quality of the proposed work. In this concern some of the salient point are as follows:

1. In this thesis work we focused on the hardware implementation and semi custom VLSI design flow for MAC architecture. The proposed $BitMAC$ can be further investigated based on CMOS custom design approach

2. We have used $180nm$ and $45nm$ technology node for the analysis purpose. Recent technology node like $22nm$ and $12nm$ can also be used for the analysis.

3. The CMOS level implementation of the proposed MAC also leads to design the $BitMAC$ incorporating the in memory compute technique. In the present work the memory element is used to store the weight value, if in memory computation technique is implemented in the design using SRAM or other non volatile memory the throughput of the design can be increased.

Finally we conclude that the intended objective of designing an efficient MAC unit for DNN is successfully achieved.

Bibliography

[1] K. Siu, D. M. Stuart, M. Mahmoud, and A. Moshovos, "Memory requirements for convolutional neural network hardware accelerators," in *2018 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2018, pp. 111–121.

[2] J. Ma, R. P. Sheridan, A. Liaw, G. E. Dahl, and V. Svetnik, "Deep neural nets as a method for quantitative structure–activity relationships," *Journal of chemical information and modeling*, vol. 55, no. 2, pp. 263–274, 2015.

[3] D. Amodei, R. Anubhai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, J. Chen, M. Chrzanowski, A. Coates, G. Diamos *et al.*, "End to end speech recognition in english and mandarin," 2016.

[4] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*, 2015.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[6] K.-L. Du and M. Swamy, "Neural network circuits and parallel implementations," in *Neural Networks and Statistical Learning*. Springer, 2019, pp. 829–851.

[7] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017, pp. 65–74.

[8] Z. Li, Y.-J. Huang, and W.-C. Lin, "Fpga implementation of neuron block for artificial neural network," in *2017 International Conference on Electron Devices and Solid-State Circuits (EDSSC)*. IEEE, 2017, pp. 1–2.

[9] G. Raut, S. Rai, S. K. Vishvakarma, and A. Kumar, "Recon: Resource-efficient cordic-based neuron architecture," *IEEE Open Journal of Circuits and Systems*, vol. 2, pp. 170–181, 2021.

[10] E. Nurvitadhi, J. Sim, D. Sheffield, A. Mishra, S. Krishnan, and D. Marr, "Accelerating recurrent neural networks in analytics servers: Comparison of fpga, cpu, gpu, and asic," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2016, pp. 1–4.

[11] G. Raut, S. Rai, S. K. Vishvakarma, and A. Kumar, "A cordic based configurable activation function for ann applications," in *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2020, pp. 78–83.

[12] D. Esposito, A. G. Strollo, and M. Alioto, "Low-power approximate mac unit," in *2017 13th Conference on Ph. D. Research in Microelectronics and Electronics (PRIME)*. IEEE, 2017, pp. 81–84.

[13] T. Yang, T. Sato, and T. Ukezono, "An approximate multiply-accumulate unit with low power and reduced area," in *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2019, pp. 385–390.

[14] *Xilinx 7 Series DSP48E1 Slice https://www.xilinx.com/support/ documentation/user guides/ug479 7Series DSP48E1.pdf.*

[15] E. Wu, X. Zhang, D. Berman, I. Cho, and J. Thendean, "Compute-efficient neural-network acceleration," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2019, pp. 191–200.

[16] G. Baccelli, D. Stathis, A. Hemani, and M. Martina, "Nacu: a non-linear arithmetic unit for neural networks," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[17] H. Kim, Q. Chen, T. Yoo, T. T.-H. Kim, and B. Kim, "A 1-16b precision reconfigurable digital in-memory computing macro featuring column-mac architecture and bit-serial computation," in *ESSCIRC 2019-IEEE 45th European Solid State Circuits Conference (ESSCIRC)*. IEEE, 2019, pp. 345–348.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[19] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, and H. Arshad, "State-of-the-art in artificial neural network applications: A survey," *Heliyon*, vol. 4, no. 11, p. e00938, 2018.

[20] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, 2017, pp. 1–12.

[21] J. Heaton, "Ian goodfellow, yoshua bengio, and aaron courville: Deep learning," 2018.

[22] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos, "Stripes: Bit-serial deep neural network computing," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–12.

[23] A. Delmas, S. Sharify, P. Judd, and A. Moshovos, "Tartan: Accelerating fully-connected and convolutional layers in deep learning networks by exploiting numerical precision variability," *arXiv preprint arXiv:1707.09068*, 2017.

[24] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, V. Chandra, and H. Esmaeilzadeh, "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 764–775.

[25] K. Yugandhar, V. G. Raja, M. Tejkumar, and D. Siva, "High performance array multiplier using reversible logic structure," in *2018 International Conference on Current Trends towards Converging Technologies (ICCTCT)*. IEEE, 2018, pp. 1–5.

[26] T. Sato and T. Ukezono, "A dynamically configurable approximate array multiplier with exact mode," in *2020 5th International Conference on Computer and Communication Systems (ICCCS)*. IEEE, 2020, pp. 917–921.

[27] R. B. S. Kesava, B. L. Rao, K. B. Sindhuri, and N. U. Kumar, "Low power and area efficient wallace tree multiplier using carry select adder with binary to excess-1 converter," in *2016 Conference on Advances in Signal Processing (CASP)*. IEEE, 2016, pp. 248–253.

[28] M. Janveja and V. Niranjan, "High performance wallace tree multiplier using improved adder," *ICTACT journal on Microelectronics*, vol. 3, 2017.

[29] S. Mirzaei, A. Hosangadi, and R. Kastner, "Fpga implementation of high speed fir filters using add and shift method," in *2006 International Conference on Computer Design*. IEEE, 2006, pp. 308–313.

[30] R. Pinto and K. Shama, "Low-power modified shift-add multiplier design using parallel prefix adder," *Journal of Circuits, Systems and Computers*, vol. 28, no. 02, p. 1950019, 2019.

[31] D. A. Gudovskiy and L. Rigazio, "Shiftcnn: Generalized low-precision architecture for inference of convolutional neural networks," *arXiv preprint arXiv:1706.02393*, 2017.

[32] M. Yuvaraj, B. J. Kailath, and N. Bhaskhar, "Design of optimized mac unit using integrated vedic multiplier," in *2017 International conference on Microelectronic Devices, Circuits and Systems (ICMDCS)*. IEEE, 2017, pp. 1–6.

[33] V. K. Jha and M. S. Gupta, "Design of 16 bit low power vedic architecture using csa & uts," 2019.

[34] S. Manikandan and C. Palanisamy, "Design of an efficient binary vedic multiplier for high speed applications using vedic mathematics with bit reduction technique," *Circuits and Systems*, vol. 7, no. 9, pp. 2593–2602, 2016.

[35] "Iso/iec/ieee international standard - floating-point arithmetic," *ISO/IEC 60559:2020(E) IEEE Std 754-2019*, pp. 1–86, 2020.

[36] S. Abed, Y. Khalil, M. Modhaffar, and I. Ahmad, "High-performance low-power approximate wallace tree multiplier," *International journal of circuit theory and applications*, vol. 46, no. 12, pp. 2334–2348, 2018.

[37] H. Jiang, C. Liu, F. Lombardi, and J. Han, "Low-power approximate unsigned multipliers with configurable error recovery," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 1, pp. 189–202, 2018.

[38] N. Van Toan and J.-G. Lee, "Fpga-based multi-level approximate multipliers for high-performance error-resilient applications," *IEEE Access*, vol. 8, pp. 25 481–25 497, 2020.

[39] A. S. K. Vamsi and S. Ramesh, "An efficient design of 16 bit mac unit using vedic mathematics," in *2019 International Conference on Communication and Signal Processing (ICCSP)*. IEEE, 2019, pp. 0319–0322.

[40] M. Alçın, İ. Pehlivan, and İ. Koyuncu, "Hardware design and implementation of a novel ann-based chaotic generator in fpga," *Optik*, vol. 127, no. 13, pp. 5500–5505, 2016.

[41] S. Anwar, K. Hwang, and W. Sung, "Fixed point optimization of deep convolutional neural networks for object recognition," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2015, pp. 1131–1135.

[42] S.-Y. Chen, R.-B. Lin, H.-H. Tung, and K.-W. Lin, "Power gating design for standard-cell-like structured asics," in *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*. IEEE, 2010, pp. 514–519.

[43] A. Abba and K. Amarender, "Improved power gating technique for leakage power reduction," *International Journal of Engineering and Science*, vol. 4, no. 10, pp. 06–10, 2014.

[44] P. S. Nair, S. Koppa, and E. B. John, "A comparative analysis of coarse-grain and fine-grain power gating for fpga lookup tables," in *2009 52nd IEEE International Midwest Symposium on Circuits and Systems*. IEEE, 2009, pp. 507–510.

[45] R. Chaintreuil, R. Uno, and H. Amano, "Mcma: A modular processing elements array based low-power coarse-grained reconfigurable accelerator," in *2013 International Conference on Reconfigurable Computing and FPGAs (ReConFig)*. IEEE, 2013, pp. 1–6.

[46] A. Sathanur, L. Benini, A. Macii, E. Macii, and M. Poncino, "Row-based power-gating: A novel sleep transistor insertion methodology for leakage power optimization in nanometer cmos circuits," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 3, pp. 469–482, 2009.

[47] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

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

[49] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[50] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[51] "Synopsys design compiler," https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/design-compiler-graphical.html.

[52] “Mentor graphics eda tools,” https://eda.sw.siemens.com/en-US/.

[53] “Cadence virtuoso system design platform,” https://www.cadence.com/ko_KR/home/tools/ic-package-design-and-analysis/ic-package-design-flows/virtuoso-system-design-platform.html.

[54] “Vivado design suite by xilinx,” https://www.xilinx.com/products/design-tools/vivado.html.

[55] “Zedboard specification,” https://www.avnet.com/wps/portal/us/products/avnet-boards/avnet-board-families/zedboard/zedboard-board-family.

[56] “Block memory generator v8.3 ip user guide,” https://www.xilinx.com/support/documentation/ip_documentation/blk_mem_gen/v8_3/pg058-blk-mem-gen.pdf, 2017.

[57] “Distributed memory generator v8.0 ip product guide,” https://www.xilinx.com/support/documentation/ip_documentation/dist_mem_gen/v8_0/pg063-dist-mem-gen.pdf, 2015.

[58] “Multiplier v12.0 ip product guide,” https://www.xilinx.com/support/documentation/ip_documentation/mult_gen/v12_0/pg108-mult-gen.pdf, 2015.

[59] “Adder/subtracter v12.0 ip product guide,” https://www.xilinx.com/support/documentation/ip_documentation/mult_gen/v12_0/pg108-mult-gen.pdf, 2021.

Publications

1. **Harsh Chhajed**, Gopal Raut, Narendra Singh Dhakad, Sudheer Vishwakarma and Santosh Kumar Vishvakarma, "BitMAC: Bit-Serial Computation based Efficient Multiply-Accumulate Unit for DNN Accelerator", Circuits, System and Signal Processing (Under Minor Revision)