# In-Memory Computing Based Hardware Accelerators Advancing the Implementation of AI/ML Tasks in Edge Devices

by

Jyotishman Saikia

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved October 2023 by the
Graduate Supervisory Committee:

Chaitali Chakrabarti, Co-Chair
Jae-sun Seo, Co-Chair
Yu Cao
Deliang Fan

ARIZONA STATE UNIVERSITY

December 2023

ABSTRACT

Artificial Intelligence (AI) and Machine Learning (ML) techniques have come a long way since their inception and have been used to build intelligent systems for a wide range of applications in everyday life. However they are very computation intensive and require transfer of large volume of data from memory to the computation units. This memory access time constitute significant part of the computational latency and a performance bottleneck. To address this limitation and the ever-growing demand for implementation in hand-held and edge-devices, In-memory computing (IMC) based AI/ML hardware accelerators have emerged.

First, the dissertation presents an IMC static random access memory (SRAM) based hardware modeling and optimization framework. A unified systematic study closely models the IMC hardware, and investigates how a number of design variables and non-idealities (e.g. device mismatch and ADC quantization) affect the Deep Neural Network (DNN) accuracy of the IMC design. The framework allows co-optimized selection of different design variables accounting for sources of noise in IMC hardware and robust implementation of a high accuracy DNN.

Next, it presents a $k$NN hardware accelerator in 65nm Complementary Metal-Oxide-Semiconductor (CMOS) technology. The accelerator combines an IMC SRAM that is developed for binarized deep neural networks and other digital hardware that performs top-$k$ sorting. The simulated $k$ Nearest Neighbor accelerator design processes up to 17.9 million query vectors per second while consuming 11.8 mW, demonstrating $>4.8\times$ energy-efficiency improvement over prior works.

This dissertation also presents a novel floating-point precision IMC (FP-IMC) macro with a hybrid architecture that configurably supports two Floating Point (FP) precisions. Implementing FP precision MAC has been a challenge owing to

its complexity. The design is implemented on 28nm CMOS, and taped-out on chip demonstrating 12.1 TFLOPS/W and 66.1 TFLOPS/W for 8-bit Floating Point (FP8) and Block Floating point (BF8) respectively.

Finally, another iteration of the FP design is presented that is modeled to support multiple precision modes from FP8 up to FP32. Two approaches to the architectural design were compared illustrating the throughput-area overhead trade-off. The simulated design shows a 2.1 $\times$ normalized energy-efficiency compared to the on-chip implementation of the FP-IMC.

ACKNOWLEDGMENTS

I would like to take this opportunity to thank my advisor and committee co-chair, Dr. Jae-sun Seo, for his priceless mentoring and his crucial role in developing my research intuition and paper organization. He has been a great source of motivation and encouragement during my pursuit of this degree. I appreciate the insights and ideas provided at each juncture to guide my research and help realise its potential. It has been a great pleasure to work with him.

I want to extend my gratitude to Dr. Chaitali Chakrabarti, Dr. Yu Cao, and Dr. Deliang Fan for their guidance in my research projects, comments and suggestions on my dissertation and for sparing the time to serve as committee members.

I am indebted to my colleagues for their cooperation and assistance in the successful realization of this dissertation.

Finally, I want to thank my family who have been a source of support and motivation throughout this difficult process.

TABLE OF CONTENTS

CHAPTER Page

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

# Chapter 1

# INTRODUCTION

Efforts to build intelligent machines has been an endeavour since the computers were invented. Machine learning (ML) covers a wide range of techniques that have been developed over the years to allow machines to learn from raw inputs to acquire some form of intelligence to solve problems in wide variety of applications. Machine learning techniques have been used in fields such as computer vision, speech recognition, natural language processing, machine translation, bio-informatics, medical prognosis/ diagnosis, computational chemistry, drug design, medical image processing, weather forecasting, material science, gaming programs etc. These techniques can be used for drawing any inference in computation tasks and in decisions making.

Various statistical techniques and heuristic based classifiers have been developed for machines to learn from raw data to acquire intelligence on application domains so that machines can draw inferences and make decisions. The development of another class of machine learning techniques called Artificial Neural Network (ANN) was inspired by biological neural networks that constitute the human brain. One sub class of ANNs called the Deep Neural Network (DNN) is built as a multi-layered network. A DNN progressively extracts aspects of an application domain from input data at different hierarchical abstraction levels at the corresponding levels of the network hierarchy. In the past decade DNN based ML techniques have shown tremendous success in providing solutions to different application domains. In many cases these ML based systems have performed as well or even better than human experts. DNNs

based ML learning techniques are therefore expected to play significant role in almost every sphere of human activity in the years to come.

With the development of computing and communication technology the past couple of decades have seen wide proliferation of handheld and mobile devices among the masses. The computational power built into these devices and the communication bandwidth provided have grown rapidly over the years enhancing their scope of use tremendously and have transformed them into mobile computing device at the network edge. Some literature refer to these as Mobile Edge Devices (or Edge Devices) in the Mobile Edge Computing (MEC) environment. They are being used for providing healthcare services, financial services, e-governance, entertainment, all forms of communication (audio, video, text) services etc. and also for carrying out professional works remotely etc. It is only logical that these edge devices be provided with ML capabilities so that the benefits of ML get extended to the masses.

The machine learning based solutions are however very compute intensive and involve handling of large volumes of data. Therefore, these require very high processing speed and large memory space. The conventional computational model requires fetching of the data from memory for the processor to carry out the computations required for the ML tasks. These involve massive number of data movement operations between the processor and the memory module. Further, the memory operations have always been significantly slower than the processors. Therefore, the memory becomes the bottleneck in the processing performance for computing devices.

Various techniques have been adopted to overcome this memory bottleneck problem. One approach to this is to use hardware accelerators that carry out certain ML tasks. These accelerators are provided with memory to hold the data to be processed by the ML tasks. This relieves the system/memory bus from the high data traffic due to the

ML tasks. One time block data transfer is done to the accelerator module to carry out an ML task. One way to implement such hardware accelerators that has gained considerable attention is In-Memory Computing (IMC) where the ML computational capabilities are built into the memory hardware of the accelerator module. As there is no requirement of a separate processing unit (PU) within the accelerator module in the IMC approach, the large latency involving memory-PU data transfers do not arise here.

The hardware accelerators for ML tasks, that involve large volume data processing, are often developed as hardware macros. These macros can be embedded in device chips that require performing of the ML task.

Viewing at it from a physical design perspective, Moore's law predicted that the number of transistors on chip would double every two years, in 1965. Until the recent decades, the trend followed this estimation, but has slowed down since. When packing the transistors closely, the heat dissipated increases. The cooling technology to mitigate this heat has not kept up with the amount of heat dissipated, resulting in hotter chips running into reliability issues. As the technology advances have slowed down in the recent years, so has the scalability and processing speed. We are facing a virtual wall in the path of progress using traditional hardware technology. This calls for other approaches to improving the performance of our systems, like better memory storage, bio-inspired computing and In-memory Computing (IMC).

## 1.1 In-Memory Computing

Technology has advanced to a point where, the access memory time is a highly significant part of the system latency. In conventional digital implementation, the

memory is segregated from the computing engine. But for IMC implementation, the computation engine is embedded inside the memory, resulting in a large reduction in memory data transfer and thus latency. As the computing hardware are attached to the memory cells, the computations can be readily performed in parallel, where data parallelism exists, thus achieving massive gain in computing speed. Additionally, it lends itself well to scalability which is a matter of adding more memory tiles along with associated computing hardware.

There are some challenges to IMC in the form of analog variability, DNN accuracy degradation and scalability concerns and mapping generalized AI workloads. SRAM based IMC have been more popular as it is more robust and lends well to large-scale integration.

## 1.2 Motivation

The primary motivations for this research were to have a better understanding of the characteristics certain IMC cell technologies and to develop IMC based hardware accelerator macros for ML tasks that are suitable for implementation in mobile edge devices in MEC environment. These mobile edge devices are small in size and are battery driven. Therefore, the accelerators have to be space and energy efficient. These devices are also likely to be used in real time applications. Therefore, to be computationally relevant these also should be able to carry out significant ML tasks within application specific deadlines. Adequate accuracy and reliability are other essential requirements. The critical performance parameters of an accelerator such as latency, energy efficiency, accuracy, silicon space efficiency, throughput and reliability. The device technology chosen to implement the hardware affects these significantly.

Proper modelling and characterization of the device technology in the context of the ML technique used is therefore important. Majority of the existing IMC based hardware accelerators use analog device technology for the computations. These suffer from limitations of accuracy. Fully digital implementation of the computations can provide higher levels of accuracy. It also can provide the option of trade off on level of accuracy against speed and packing density. This motivates us to work on fully digital IMC implementation of the ML hardware accelerators.

In conventional IMC based hardware two of the functionalities that have wide utility in machine learning algorithms are:

1. $K$ Nearest Neighbour (kNN) computation: It is widely used in classifiers because of its simplicity and versatility
2. High accuracy floating point computation: Millions of multiply and accumulate operations are required to be performed in every plane of a DNN. For high accuracy these operations needs to be done in a FP format. IMC implementation provides the scope for implementation of these large number of operations in parallel through appropriate replication of the hardware. The specific FP format and accuracy level required depends on the applications. The hardware accelerator for FP computations configurability of FP format is desirable.

AI/ML techniques require a large storage in the form of millions of weights and billions of MAC operations for an inference. These involve repetitive memory access. It is a daunting task to implement these complex AI/ML algorithms in hardware.

To compare the AI/ML hardware accelerators, energy-per-inference is the recognized metric and has been commonly used in literature (Seo *et al.* (2022)). The energy-per-inference is essentially:

- The number of operations per inference
- The energy per operation

The first is fine-tuned by exploiting the algorithm model characteristics, while the latter is associated with the process technology implementation.

## 1.3 Objectives

A set of generalized objectives were set.

- **High throughput:** To make an effort to improve the throughput and meet the ever-increasing demand for higher processing speed, more so in edge-devices.
- **Low Power Solutions:** With the additional architectural area overhead to improve throughput, the power can scale up dramatically. A conscious effort has to be made to employ solutions to lowering power consumption.
- **Algorithm-level Optimization:** Targeting specific algorithms allows for exploiting specific algorithmic characteristics, rather than going for a general all application approaches.
- **Novel Design:** An effort to approach under-explored areas of research and find a solution to available problems in niche fields.

Based on the studies carried out and the analysis of the requirements made the specific objectives of this research were set as follows:

1. To develop tools to characterize the effects of non-idealities and network architecture on DNN accuracy. This is to help optimize the choice of design parameters for developing DNNs with robust noise characteristics, accuracy and energy efficiency.

2. To develop low latency and energy-efficient hardware accelerators for the functionality of $k$NN classifier. That is often required in AI/ML accelerators. Exploiting certain property of IMC SRAM-XNOR device row accumulative output.
3. To provide efficient IMC solutions to the complex problem of implementing the layers of DNN with its characteristic multiplication and accumulation (MAC) operations in FP formats with support for a wide range of precision modes.

## 1.4 Contributions

The key contributions of this thesis to advance the DNN implementation in edge devices include:

- A SRAM-based IMC hardware modeling and optimization framework developed in python. It is a unified systematic study that closely models IMC hardware and investigates how a set of design variables (e.g Batch size, ADC precision) and non-idealities (e.g. device mismatch and ADC offset error) that affect the DNN accuracy in an IMC implementation. The characterization of this relationship between the non-idealities and the DNN accuracy is used to develop a more noise-robust Capacitive IMC Chip.
- A kNN accelerator that combines in-memory computing SRAM developed for binarized neural networks and a top-k sorting digital hardware. The digital sorter design is simulated for 65nm tech, while using the on-chip data available for the distance computation engine. The kNN accelerator processes up to 17.9 million query vectors per second while consuming 11.8 mW, demonstrating >4.8X energy improvement over prior works.
- A novel floating-point precision IMC (FP-IMC) macro implementing a 8x8 block

of a DNN layer . This design supports configurability of both conventional floating point (FP8) and block floating point (BF8) formats for the Multiply-and-Accumulate operations. Its implementation on chip in 28nm technology demonstrated an operating frequency of 650 MHz and 2.4GHz for the FP8 and BF8 modes respectively. It also provided significant energy-efficiency over non-IMC implementations and other FP precision works.

- An improvement to the FP-IMC macro employing higher precision modes including FP8, BF16, FP16, TF-32, PXR24, AMD24 and FP32. The design also incorporates architectural improvements including a modified Adder Tree scheme and exponent handling for better energy-efficiency. This work is able to show an additional 2.1× normalized energy-efficiency over the FP-IMC.

## 1.5 Thesis Organization

The rest of this thesis is organized as follows:

- **Chapter 2** gives some background into In-Memory computing and DNNs.
- **Chapter 3** shares a python based framework to understand the effects of IMC non-idealities on DNN training accuracies.
- **Chapter 4** presents a kNN accelerator taking advantage of the characteristics of the IMC SRAM.
- **Chapter 5** presents a novel IMC accelerator, FP-IMC, implementing MAC operations in configurable FP formats in blocks of DNN planes
- **Chapter 6** presents the architectural changes to the FP-IMC accelerator macro to support higher precision modes and improve normalized energy-efficiency.

- **Chapter 7** concludes the dissertation by sharing insights gained from experience and potential future work.

# Chapter 2

# BACKGROUND

In this chapter we discuss the structural organization of the Deep Neural Networks (DNN) and some of the memory cell and cell group level architectures and techniques used in In-memory Computing (IMC) that are core to the works presented in this thesis.

## 2.1 Deep Neural Networks (DNN)

The Deep Neural Networks (DNNs) constitute a subclass of Artificial Neural Networks. To give more context to DNNs, let's look at a simple ANN shown in Figure 1. The 3-layer Fully connected(FC) ANN shown in figure has a 3 node input layer, a 5-node hidden layer and an 2-node output layer. For a complex problem there can be several hidden layers with more nodes per layer.

There are directed edges connecting the nodes of one layer to the next, with an associated weight. For instance, in the example, the edge between Input Layer node k, $L_{0,k}$ and Hidden Layer node j, $L_{1,j}$ has a weight, $W_{1,j,k}$. The activation of the j-th node in layer i, can then be computed as:

$$L_{i,j} = \sum_{k=0}^{K-1} L_{(i-1),k} \times W_{i,j,k} \tag{2.1}$$

Here, k=0,..,(K-1) are the nodes in the previous layer,$L_{i-1}$. The computation in equation 2.1 is essentially a multiply-and-accumulate (MAC).

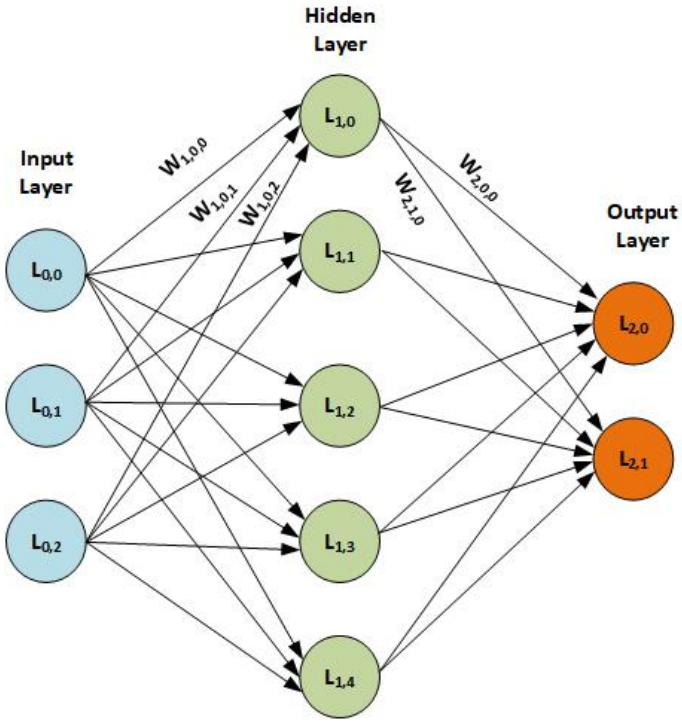



Figure 1. A Basic 3-layer Artificial Neural Network (ANN)

All the nodes in a layer, in the example network, are connected to all the nodes in the next layer with edges, these are referred to as Fully Connected (FC) layers. However, the connectivity between the layers can be partial too.

The nodes of the output layer $L_{2,0}$ and $L_{2,1}$, can be classes in the case of a classifier ANN or a property value in the case of a housing market regression tool. For the classifier ANN, the $L_{2,0}$ and $L_{2,1}$ will be the associated probability value for the two classes. Based on the input features of Image 'Input', $L_{0,0}$ ... $L_{0,N}$, it is assigned to the class with the higher priority between $L_{2,0}$ and $L_{2,1}$. During training, the weights are tweaked as per the training dataset image input features to return the correct class.

The accuracy of these ANNs also relies profoundly on the extracted input features (or abstractions). To extract these features DNNs rely on convolutional layers. In Figure 2, a basic convolutional neural network CNN is shown where convolution operations are used to extract the features and FC layers are used to classify. In the

convolutional layer, sets of weight kernels are used to extract several feature maps. This convolution of input image array and weight kernels also involves a Multiply-and-accumulate computation. Some pooling and flattening techniques are used to translate these feature maps into the inputs to the FC layer where the classification occurs.

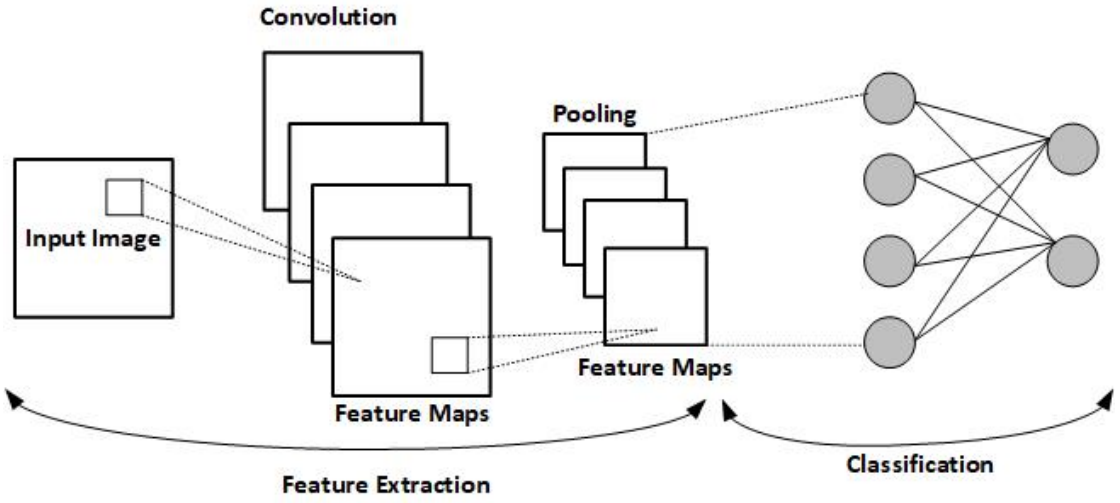



Figure 2. A Basic Convolutional Neural Network (CNN)

As eluded to previously, the extraction of intricate features from the input images is perhaps the most important aspect of the DNNs. Usually, the extraction of the features needs to be carried out at different hierarchical levels of the features. The number of the hierarchical levels depends upon the level of details that need to be considered. This also decides the number of convolutional layers required in the DNN. In the Figure 3, we show a ResNet-18 DNN for CIFAR-10. Just one of the 18 layers in the ResNet-18 is a FC layer, and rest of them are convolutional layers. There are examples of DNNs. e.g. ResNet-150, that use much larger number of layers.

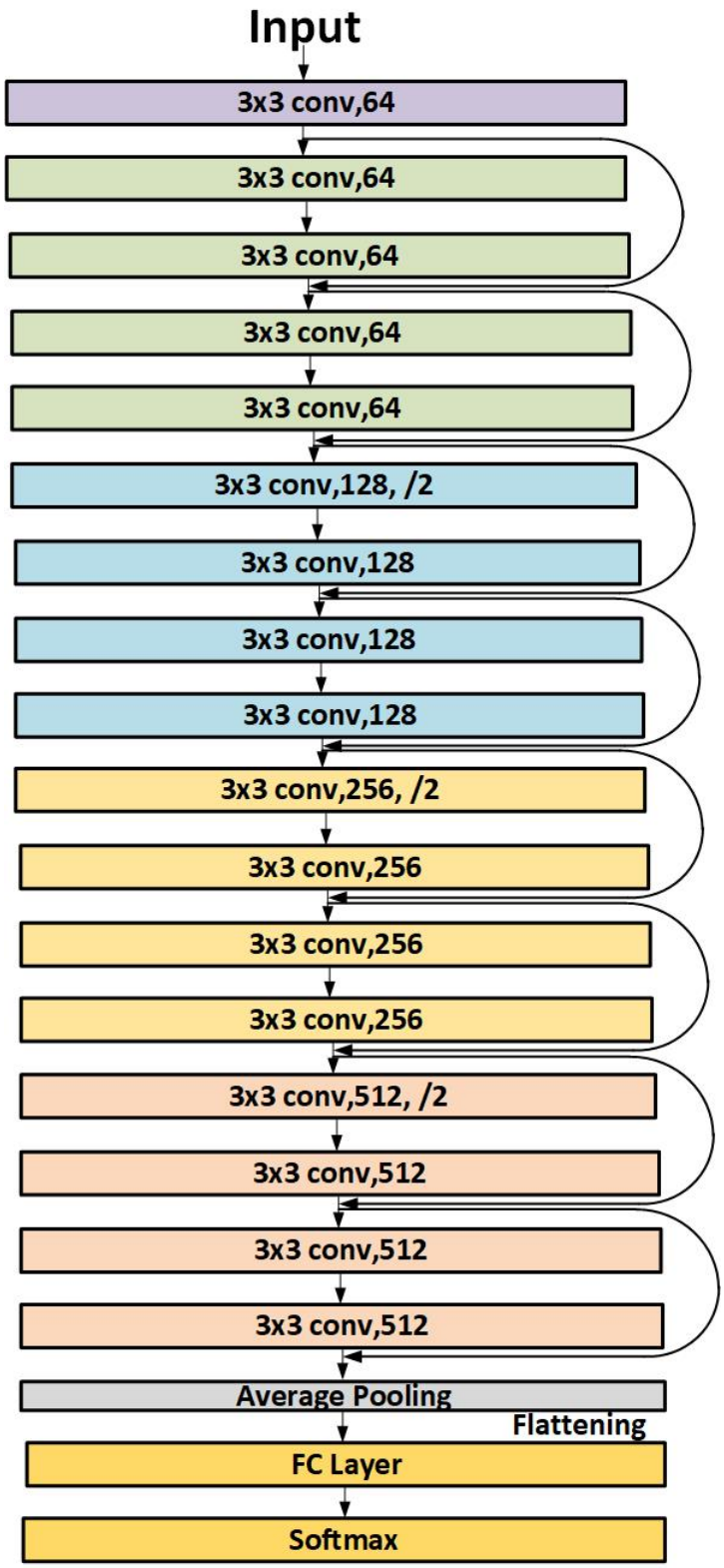



Figure 3. The Architecture of the ResNet-18 DNN on CIFAR-10 Dataset

## 2.2 In-Memory Computing

In-memory computing devices popularly use Static RAM (SRAM) instead of Dynamic RAM due to the higher speed that can be achieved in SRAM and to avoid the complexities due to the periodic refresh requirements in DRAM. The 6T SRAM cell (in Figure 4) is the popular choice for bit storage device in SRAM due to its robustness and the high packing density that it allows. Further, it also lends itself well to integration of computational features at the individual cell level as well as in cell groups.

In the 6T SRAM, shown in figure, the T1-T4 transistors form a back-to-back connected pair of inverters to hold the weight. While the T5-T6 transistors are used to feed in the weights to be stored, via the Bit Line (BL), by toggling the Wordline (WL).

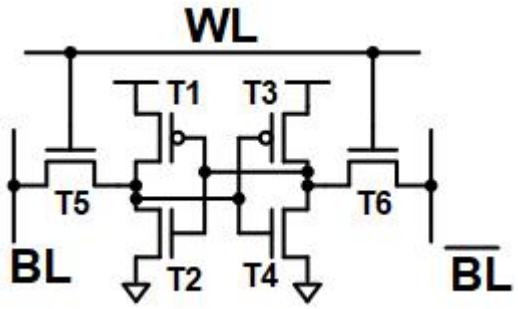



Figure 4. The Transistor Level Description of 6T SRAM

As discussed in Section 2.1, the forms of computations most commonly required in machine learning and neural networks are summing and multiplication operations in addition to the normal logical operations.

The summing operation in IMC SRAM is usually achieved in analog form through accumulation of either current on a line, voltages across resistors or charges in a

capacitor and sensing the sum. Some form of ADC is used to get the sum in digital form. In certain cases, the summing is done digitally. While analog summing is faster and the circuit occupies lesser space, the digital operation provides better accuracy and noise immunity.

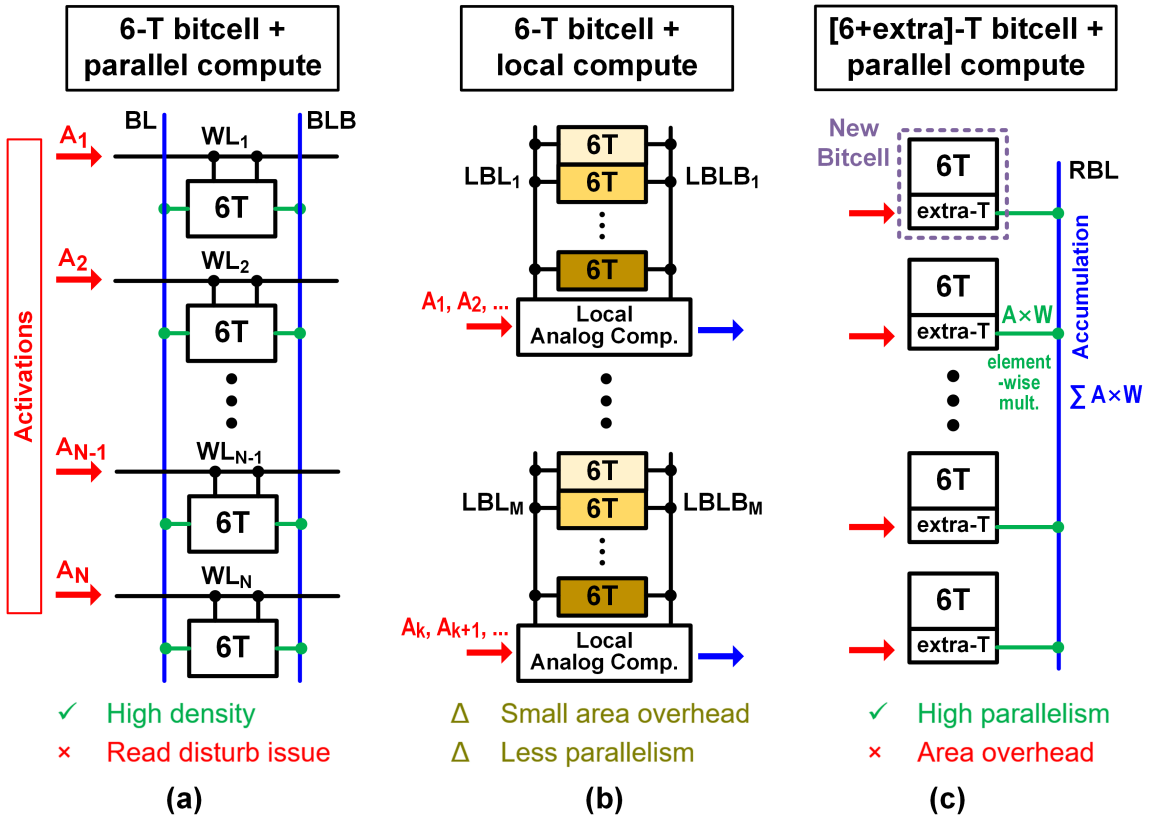



Figure 5. Categorization of the SRAM IMC schemes

The Figure 5 shows the different analog SRAM IMC schemes. Early analog IMC SRAM works, shown in (a), accumulated over a Bit Line (BL) which resulted in a read disturb issue exacerbated with scaling. Then, we saw the introduction of grouped bitcells, shown in (b), with a local compute which alleviated the read-disturb issue. This offers more parallelism for an area overhead. The scheme shown in (c), goes

one step further introducing a local compute for each bitcell resulting in a very high parallelism and elimination of read-disturb for a larger area overhead.

Prominently there are two forms of analog SRAM IMCs: resistive and capacitive. Resistive IMC works use resistive pull up/down to perform bitcell multiplication. When observing the transistor characteristics across a wide voltage range, it exhibits a non-linear curve. Capacitive IMC works employ charge sharing or capacitive coupling by using an additional capacitor per bitcell to perform the MAC. The Read Bit Line (RBL) transfer curve shows a more linear curve, as the variability of capacitors is less than that of transistors. These analog SRAM IMCs exhibit a low SNR, along with which comes intra-chip and inter-chip variability.

### 2.2.1 Capacitive IMC Designs

In the two works, shown in Figure 6, capacitive computing use is demonstrated. For the bitwise multiplication in the targeted binarized Neural Network (BNN), both the works use -1/+1 activation/weight. In the first work Valavi *et al.* (2019), the capacitor is either charged when product of the activation and weight is +1 or discharged, when the product is -1. In the second work, Jiang *et al.* (2020) the intermediate node $V_C$ is driven by the activation from the MAC word line (MWL). The weights are stored inside the 6-T SRAM. Once that is done, it is coupled through a series capacitance, $C_C$.

In the first work, some of the cells have been fully charged while others are discharged. By connecting the switch between the capacitors and the BitLine (BL), charge sharing happens and the overall voltage is an averaged analog voltage that represents the MAC value.

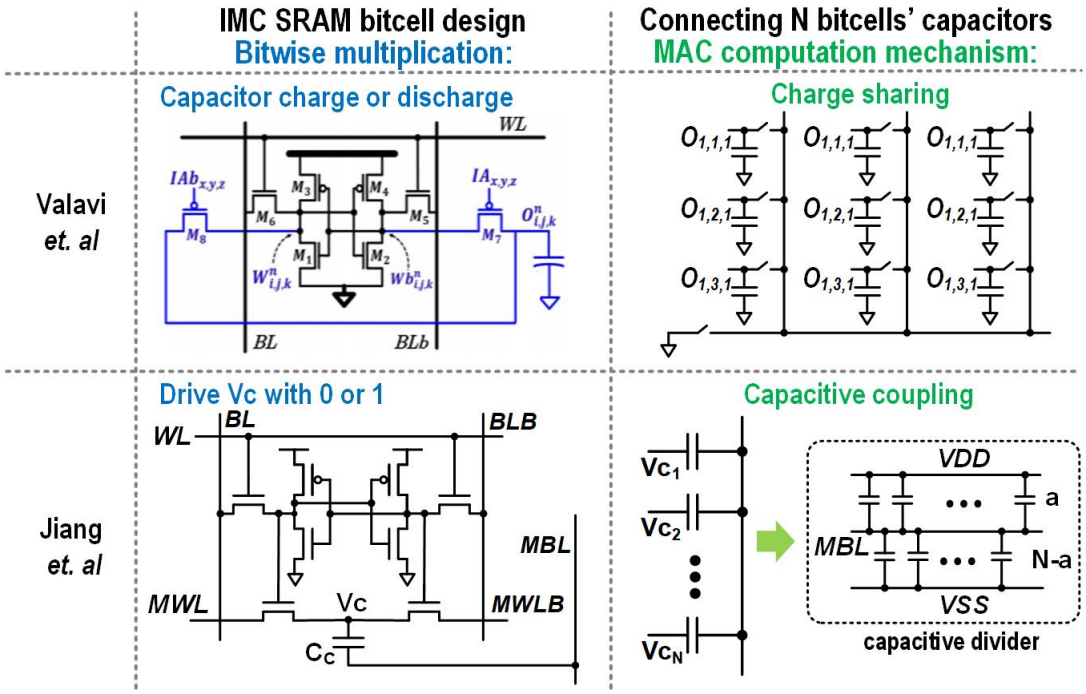



Figure 6. The Capacitive IMC Circuits and Operation (Left) the IMC SRAM Bitcell Design, (Right) N Bitcells connected to the same RBL

In the second work, the N cells are connected to the same MAC bitline (MBL) and that is done by $V_C$ driving the series capacitor $C_C$, through the MAC bitline. This basically achieves a capacitive divider where some number of cells that have driven the $V_C$ to low, and some number of cells in the column that have driven $V_C$ to high. And depending on the ratio, MBL will also result in an analog voltage MAC.

### 2.2.2 Resistive IMC Designs

We use Dong *et al.* (2020) and Yin *et al.* (2020) as reference works for the Resistive IMC Designs. The bit-wise multiplication of the weights and the activation are performed in the individual bitcells shown in Figure 7,(Left). For Dong *et al.* (2020) work, the inputs and weights can be either '0' or '1', and the multiplication is performed by pulling down the RBL when the input and weights are both '1' or both '0'. In

the XNOR-SRAM work in Yin *et al.* (2020), additional transistors are employed to perform the XNOR operation in BNNs, whereby when inputs and weights are both '1' or '0' the RBL will be pulled up.

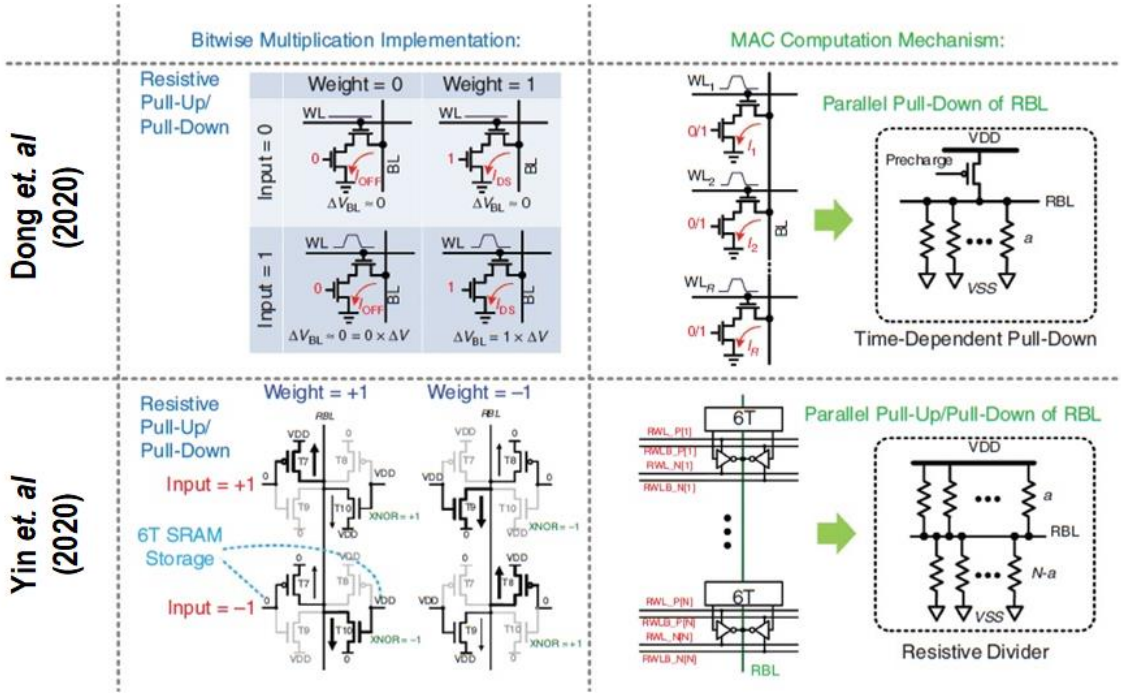


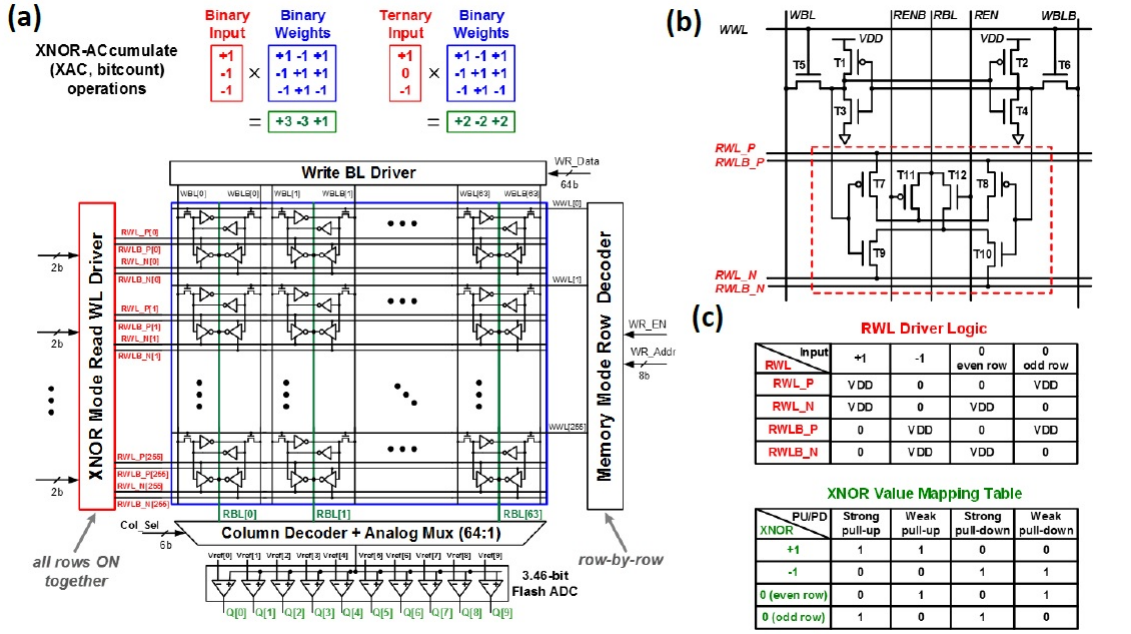

Figure 7. The Resistive IMC Circuits and Operation (Left) IMC SRAM Bitcell Design, (Right) N Bitcells connected to the same RBL

In Figure 7(Right),the accumulation of the column which is inherently analog is shown. N rows of the column are activated at once to interact with the same RBL over which the accumulation takes place. In Dong *et al.* (2020), the RBL is precharged to VDD, and discharge based on the number of bitcells that are pulling down. This MAC value from the RBL is evaluated over a period of time, making the evaluation some-what time sensitive. InYin *et al.* (2020), a resistive divider is formed based on how many bitcells pull up or down. Based on that a crow-bar current flows through the RBL which can be evaluated to obtain the MAC value.

## 2.3 The XNOR SRAM

Yin *et al.* (2020), shares an in-memory computing SRAM cell, the XNOR-SRAM, performing a XNOR and accumulate (XAC) operations that is predominantly used in Binary Neural Networks (BNNs) to achieve high energy-efficiency and throughput.

Figure 8(a), presents the 256-by-64 memory array architecture with the peripheral circuitries that can be mapped onto a Fully Connected (FC) layer or a convolutional layer in Convolutional Neural Networks (CNNs) and Multi Layer Perceptrons (MLPs).




| RWL \ Input | +1 | -1 | 0 even row | 0 odd row |
|---|---|---|---|---|
| RWL_P | VDD | 0 | 0 | VDD |
| RWL_N | VDD | 0 | VDD | 0 |
| RWLB_P | 0 | VDD | 0 | VDD |
| RWLB_N | 0 | VDD | VDD | 0 |

| XNOR \ PU/PD | Strong pull-up | Weak pull-up | Strong pull-down | Weak pull-down |
|---|---|---|---|---|
| +1 | 1 | 1 | 0 | 0 |
| -1 | 0 | 0 | 1 | 1 |
| 0 (even row) | 0 | 1 | 0 | 1 |
| 0 (odd row) | 1 | 0 | 1 | 0 |

Figure 8. (A) XNOR-SRAM Yin *et al.* (2020) Architecture Mapping the XNOR-and-Accumulate (XAC) Operations (B) Bitcell Schematic with 6 Additional Ts on top of the 6T SRAM (C) XAC Operation with Ternary Activations and Binary Weights

Figure 8(b), shows the 12T SRAM bit cell employed in the architecture. The four transistor T7 to T10 are used in addition to the six transistors (T1-T6) constituting the 6T SRAM, to perform the XNOR operation. T11 and T12 are used to power gate the pull-up/down circuits.

The XNOR-supports binary weights (+1/-1) and binary (+1/-1 or +1/0) as well as ternary (+1/0/-1) inputs. Figure 8(c), shows how the primary Read Word Line (RWL) converts the input to four RWLs to appropriately feed in the ternary activation logic.

The XNOR-SRAM was taped-out on-chip in 65nm CMOS and obtains 81.28pJ and 178ns for 64 operations with 256-input XAC at 0.6 V. This boils down to 2.48fJ per operation and 403 GOPS/W. The DNN implementation with the XNOR-SRAM achieved 85.7% (@90.7% baseline) for CIFAR-10 and 98.3% (@98.8% baseline ) for MNIST. Finally, the XNOR-SRAM also shares the ADC amongst the 64 columns with its column-by-column operation.

The key characteristic of the XNOR-SRAM that is of note to this thesis work is the distance computation properties that is integrated into the $k$NN work. The XNOR operation of the activation and weight bits, gives information of the similarity between bits of same significance. The XAC value gives concrete information on the similarity between the activation and the weight vectors. This is similar to Hamming distance where a XOR operation is performed on the two vectors. More specific details regarding this distance metric and how it is obtained will be discussed in chapter 4.

## 2.4 Summary:

In this chapter we have discussed the structure of DNNs and observed that the Multiply and Accumulate (MAC) operations dominate the computations in the DNN planes. We then discussed the different bit cell and groups of bit cell level constructions of SRAM IMC devices that achieve the MAC computations within the memory devices.

An understanding of the comparative performance of the different approaches to the construction these devices is also obtained.

# Chapter 3

# MODELING AND OPTIMIZATION OF SRAM-BASED IN-MEMORY COMPUTING HARDWARE DESIGN

## 3.1 Introduction

There are a number of design parameters to consider when implementing a DNN hardware. The multi-dimensional design optimization opportunities has however been been under-explored in the literature. In this work, we make a study and develop a python based simulation framework to optimize the DNN accuracy by considering a number of design parameters while accounting for device non-idealities. The study shows that in order to obtain a high DNN accuracy, all of the design parameters studied here need to be chosen carefully based on the DNN under implementation. The framework allows the designer to choose an optimal set of parameter values.

The key challenge in an analog IMC SRAM implementation is the analog variability that can lead to a significant DNN accuracy degradation. There are a number of design parameters to consider such as the number of active rows, the partial sum quantization scheme and the choice of ADC precision. These parameters have a significant role to play in the DNN accuracy as well as energy-efficiency. While state-of-the-art works may have optimized their prototype works in these areas, the optimizations are not reported in a detail. So, there arises a need for a comprehensive study of the design parameters and analog variability associated with the analog IMC SRAM hardware on the DNN accuracy and the design's energy-efficiency.

As stated above, there have been a couple of works covering this area of study

but not in great detail. Jia *et al.* (2020) reported that with more active rows , the signal-to-quantization-noise-ratio (SQNR) worsens, but the relationship between the SQNR and the DNN accuracy is not delved into. In Kang *et al.* (2020), a study was conducted in a simple ML algorithm for a small number of active rows. And the discussion was around the energy-accuracy trade-off and the effect of variation on the RBL voltage swing.

In this study, we consider multiple design parameters,

1. Number of active rows
2. Partial Sum Quantization Schemes
3. ADC precision
4. ADC offset errors
5. Bitcell variation on RBL voltage

While the first three parameters are variable choices tied to algorithmic parameters that we can tweak around to determine an optimal operating point. The latter two are analog variability noises that we expect to observe in the actual hardware on-chip implementation. The analysis was performed for binary, 2-bit and 4-bit activation/weight precision, for CIFAR-10 and ImageNet datasets. The code for this work is publicly available in the repository: https://github.com/Jsaikia47/IMC-SRAM-Modeling .

This work was a collaborative effort with a group in Columbia University focusing on the implementation of the C3SRAM, a capacitive based IMC hardware. This collaboration effort was to use the available observed on-chip noise in Jiang *et al.* (2020) (from the same group), to optimize the future implementations of the hardware accelerators using the same C3SRAM bitcell in Zhang *et al.* (2022) (from the same group).

## 3.2 IMC Hardware Modeling Framework

For this work, a python based IMC hardware modeling framework was developed to model multiple variations/noises which include ADC quantization, ADC offset and capacitance mismatch. The key features being the bit-by-bit computation aspect of IMC, specifically the fully connected layers and convolutional layers. Other DNN based operations such as pooling, batch normalization are performed digitally so as not to affect the DNN accuracy. Resnet-18 on datasets CIFAR-10 and ImageNet are used for this study.

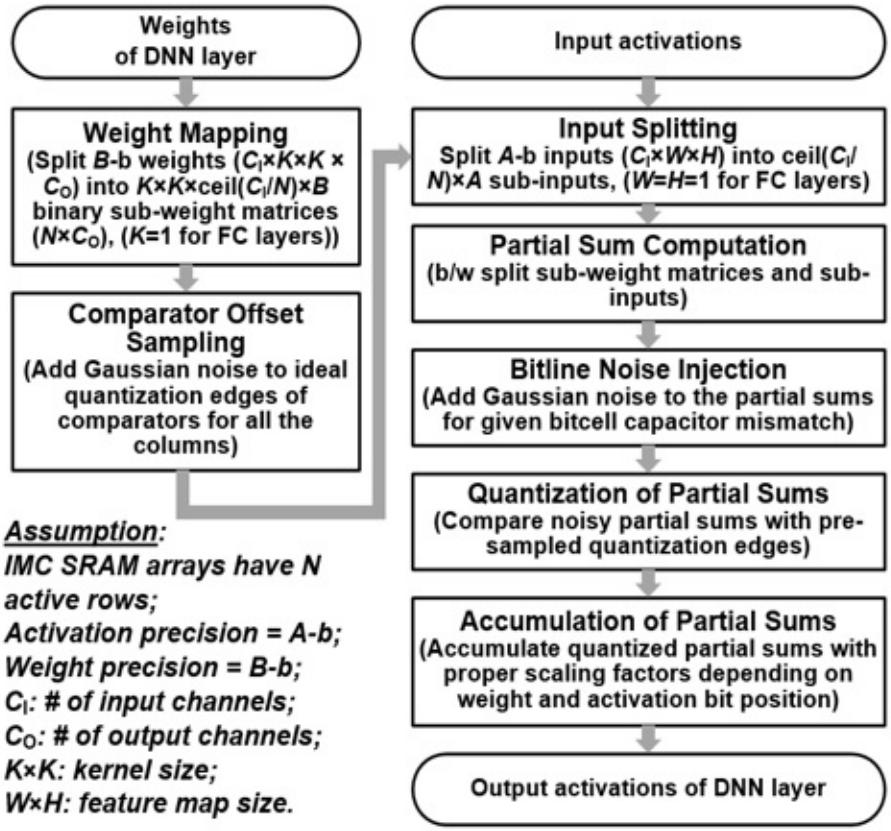



Figure 9. High-level Flow of the Python-based IMC Modeling Framework

### 3.2.1 The Python Framework

In the figure 9, the python framework of the design is implemented for a Fully Connected (FC) and a Convolutional layer. The input to the framework are the inputs and weights of the DNN layer. The work flow starts off with splitting the weight into N separate 1-bit weights, where N is the weight precision. These N 1-b values are stored in the memory. A digitized comparator offset value is modelled based on the expected variance using a Gaussian model. This value is to be added later to the partial sums of individual bit-by-bit multiplications. Similarly, the input activations are also split into N channels based on the input precision. The bit-by-bit multiplication is then performed and the comparator offset added. The partial sums are then quantized between the range of -1 to +1 into $2^N$ levels. Finally the partial sums are accumulated to obtain the output activation.

### 3.2.2 Modeling the Noise and Quantization Range Determination

As mentioned above, two noises are modeled in this modeling framework: the ADC offset error and the bitcell/capacitance variation. The bitcell capacitance variation comes from the capacitance mismatch and the ADC offset error comes from the imperfect quantization edges that lead to a quantization error.

Both of these noises are applied in some form to the RBL. The RBL voltage is calculated in a $N$ rows IMC SRAM array as:

$$V_{RBL} = \frac{\sum_{i=1}^{N} c_i \times y_i}{\sum_{i=1}^{N} c_i}, \tag{3.1}$$

Here, $y_1, \cdots, y_N$ represents the bitwise binary MAC computation results from the $N$ rows. Each of this will be either VDD or 0. The $c_1, \cdots, c_N$ representing the

capacitance values of each of the rows from the IMC SRAM array. These capacitance values are randomly sampled from a Gaussian distribution with a certain standard variation over mean ($\sigma/\mu$) value. With the equation 3.1 available to us, we calculate the standard deviation of $V_{RBL}$ and convert into unit MAC. This standard deviation is then used to model the bitcell capacitor mismatch noise on the $V_{RBL}$.

The offset value of the ADC, chosen from an estimated operating range of the ADC, is converted into unit MAC value. The ADC offset error is modeled as a Gaussian noise with standard deviation of the chosen unit MAC offset value. To adequately select the quantization scheme, the partial sum distributions of the activation values across the DNN are observed. We conduct an experiment applying quantization ranges varying in multiples of $\sigma$. The range is then clipped and a linear quantization is applied inside this range.

### 3.2.3 Evaluation of Signal-to-Quantization Noise Ratio (SQNR)

A key factor in the accuracy of the DNN implementations is the signal-to-quantization-noise-ratio (SQNR). To calculate this SQNR, we observe the noisy activation of a given layer, terming it 'X' and comparing it against the ideal activation of the layer, terming it 'signal'. This gives us the noise of the DNN the layer as 'X-signal', and the SQNR is calculates as:

$$SQNR = E[signal^2]/E[(X - signal)^2]. \tag{3.2}$$

## 3.3 Experimental Results

First, a set of isolated experiments are conducted to determine an optimal choice of design parameters without the introduction of noise. Thereafter, using these choices to characterize the impact of noises implementing ResNet-18 on CIFAR-10 and ImageNet.

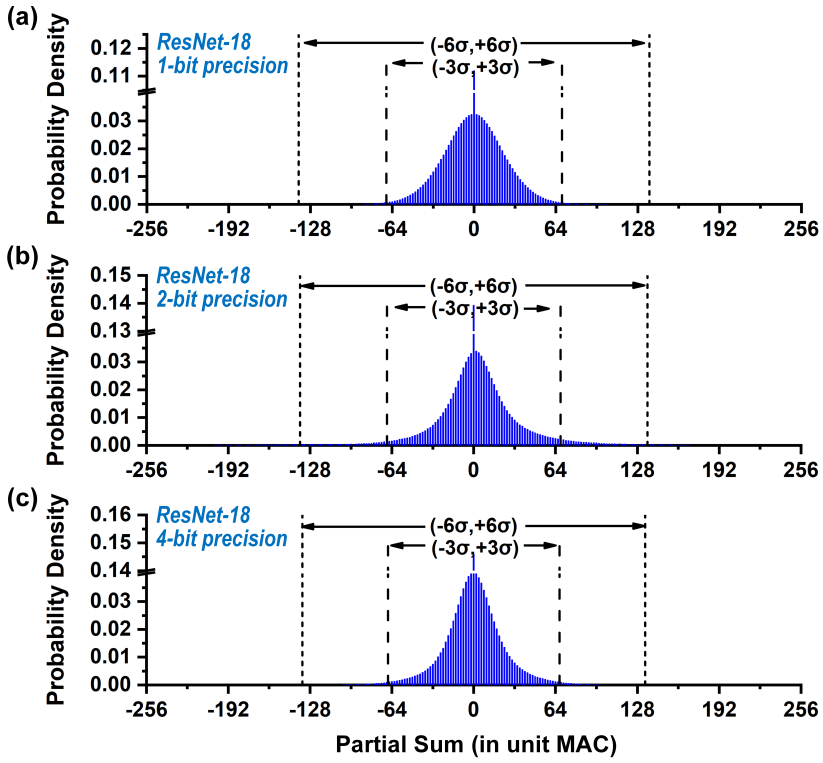



Figure 10. Column-wise Partial sum Distributions for ResNet-18 for CIFAR-10 with Different Activation/Weight Precision (a) 1-bit (b) 2-bit (c) 4-bit

### 3.3.1 Quantization Range

The first experiment revolves around determining a suitable quantization range. The partial sums are clipped beyond a quantization range and a linear quantization scheme is applied within this range. Figure 10 shows the partial sum distribution of

ResNet-18 on CIFAR-10 with activation/weight precision of 1-bit,2-bit and 4-bit. As illustrated the range -3$\sigma$ to +3$\sigma$ covers more than 99.73% of the entire distribution.

The quantization range is increased in multiples of $\sigma$ in Figure 11. Initially the error drops as a broader distribution is covered, but beyond a point the error increases again as the quantized activations are too far off the actual value. While it varies with the network precision -7$\sigma$ to +7$\sigma$ was observed to offer minimal accuracy degradation.

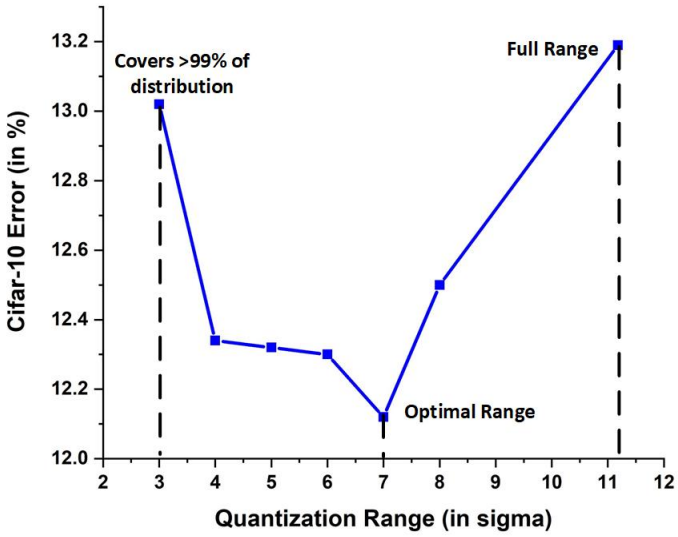



Figure 11. Effect of Quantization Range on CIFAR-10 Error

We will show in upcoming experiments that introduce noise, there are three key factors regarding the quantization range that governs the DNN accuracy:

1. A wider quantization range allows wider quantization levels. But wider quantization levels result in an inaccurate quantization of the activation.
2. A smaller quantization range is vulnerable to outliers. The points beyond the minimal and maximal quantization point of the chosen range have a severe impact on the accuracy as the magnitude of is larger.

3. A smaller quantization range also results in smaller quantization levels which are more vulnerable to noise. We will provide evidence of this in section 3.3.3.

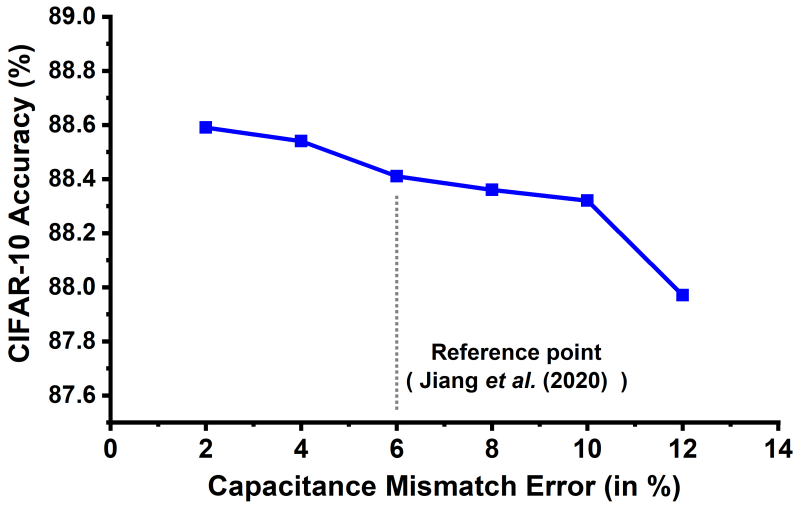



Figure 12. Effect of Capacitance Mismatch Number on DNN Accuracy. The Capacitance Mismatch Error is swept for Binary ResNet-18 for CIFAR-10 at Full Quantization Range

### 3.3.2 Capacitance Mismatch and Number of Activated Rows

For the capacitive IMC , the non-idealities include the capacitance mismatch error which is essentially the bitcell variation that impacts the RBL voltage. As detailed in section 3.3, this noise is modeled as a Gaussian noise. After averaging over multiple iterations, a trend of decreasing DNN accuracy with an increase in the capacitor mismatch error is observed as in Figure 12. In the figure the reference point at 6% capacitance mismatch error is the observed noise of the on-chip implementation work reported in Jiang *et al.* (2020)

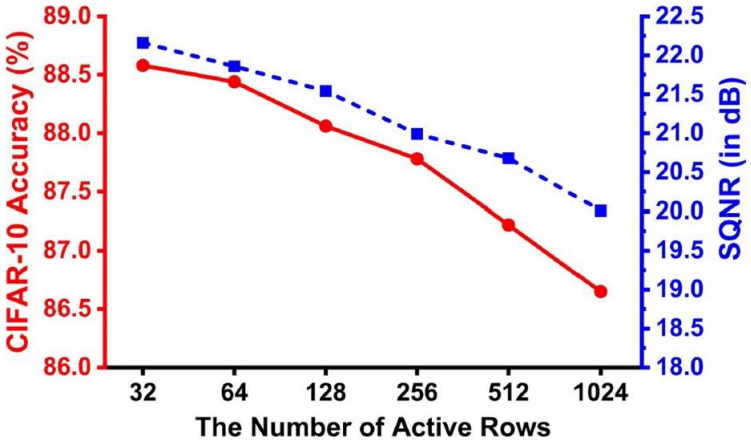



Figure 13. Effect of Number of Active Rows on DNN Accuracy. SQNR and DNN Accuracy are presented for Different Number of Activated Rows

We also conducted an experiment to characterize the impact of the number of active rows on SQNR and DNN accuracy in Figure 13. To do this we introduced the 6% capacitance mismatch and additionally, we applied a 5mV ADC offset (the upper range of the modeled Gaussian noise). As the number of rows turned on at once increased, the SQNR dropped and consequently the DNN accuracy dropped. The accuracy drops significantly for an increase in number of active rows beyond 256. As we also care about energy-efficient solution and a high throughput. Accounting for both of these factors we choose to turn on 256 rows for the upcoming experiments. But it is important to note that this knob can be turned to gain back some DNN accuracy if needed.

### 3.3.3 Experiments of ResNet-18 on CIFAR-10

In the following sets of experiments we add both the capacitance mismatch error and ADC offset error for ResNet-18 on CIFAR-10 to observe the effect on accuracy. Starting with the first set of experiments conducted at 2b activation/weight precision

to show the effect of quantization range on the CIFAR-10 accuracy, at different ADC precisions, shown in Figure 14. The quantization range is swept from 132($3\sigma$) to 512 (Full Range). In the figure, the black dotted line is the software baseline, the red plot is for non-noisy quantization only case. For the blue plot, the 6% capacitance mismatch error is introduce. And for the green and purple plot, in addition to the capacitance mismatch error, a Gaussian noise with standard deviation of 2.5 mV and 5 mW is introduced respectively.

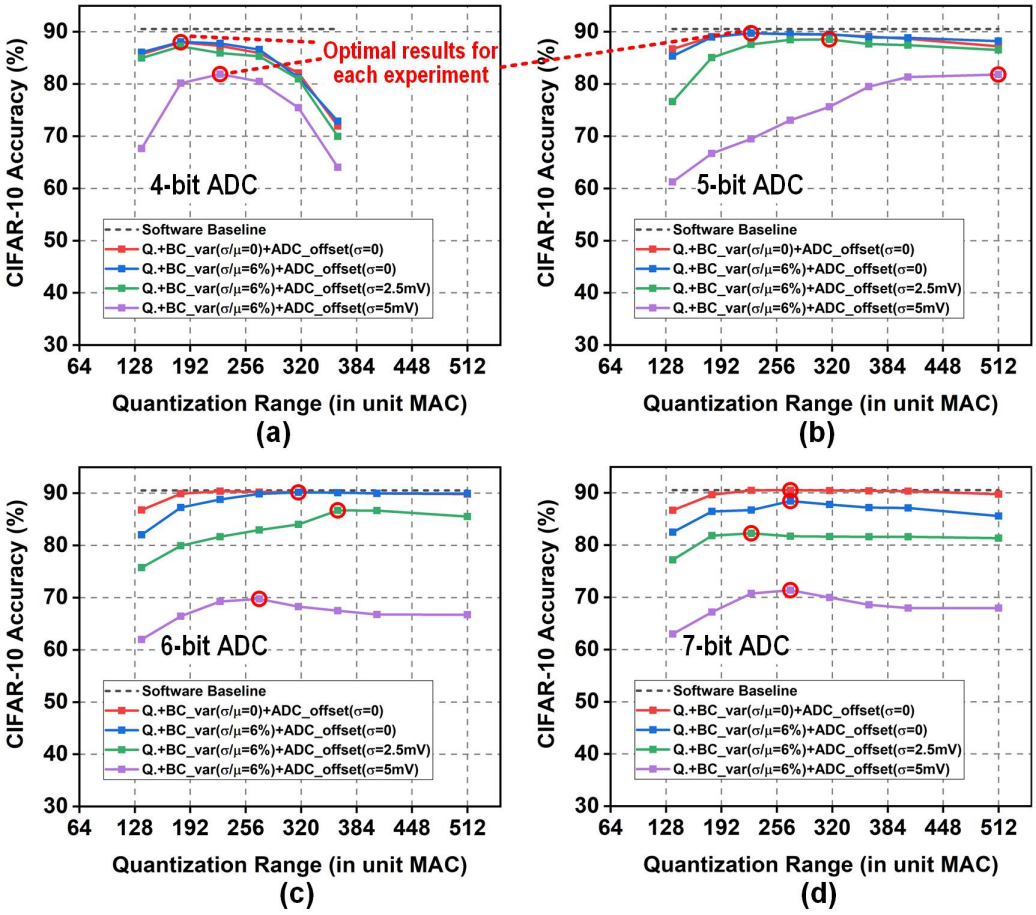



Figure 14. Detailed Simulation Results with the IMC Modeling Framework for ResNet-18 for CIFAR-10 with 2-bit/2-bit Activation/Weight Precision. For ADC Precision of (A)4-bit (B)5-bit (C)6-bit (D)7-bit, Results for Various Quantization Ranges, Bitcell Variation and ADC Offset Values.

For the 4-bit ADC, the optimal point, with the highest DNN accuracy marked in a red circle, was at a lower quantization range compared to the higher ADC precisions. This is because lower quantization ranges offer a small quantization levels for a more accurate quantized activation, while wider ranges result in "widely-off" quantized activations.

As we move from a 4-bit ADC to a 5-bit ADC, larger quantization ranges can be better explored meaning the optimal point shifts towards larger quantization ranges. For the non-noisy cases (red plots), we don't see much improvement in the accuracy. This is because the quantization range has been well explored already at 5-bit ADC. However, for the noisy cases, as we move from 5-bit ADC to 6-bit and 7-bit for a fixed quantization range, the accuracy actually drops as the finer quantization levels are more susceptible to error from ADC offset error primarily. For ADC offset error of 2.5 mV (green plot), the 5-bit ADC shows the best optimal point.

Conducting the same set of experiments for binary and 4b activation/weight precision and compiling only the optimal points across all ADC precisions we have the Figure 15. For the binary DNN, in (a), we observe the expected trend of increase in accuracy with increasing ADC precision, even for the noisy cases. And for the 4-bit DNN, in (c), similar to the 2-bit DNN case covered already, for ADC precision 6 and higher, the accuracy drops for the noisy cases.

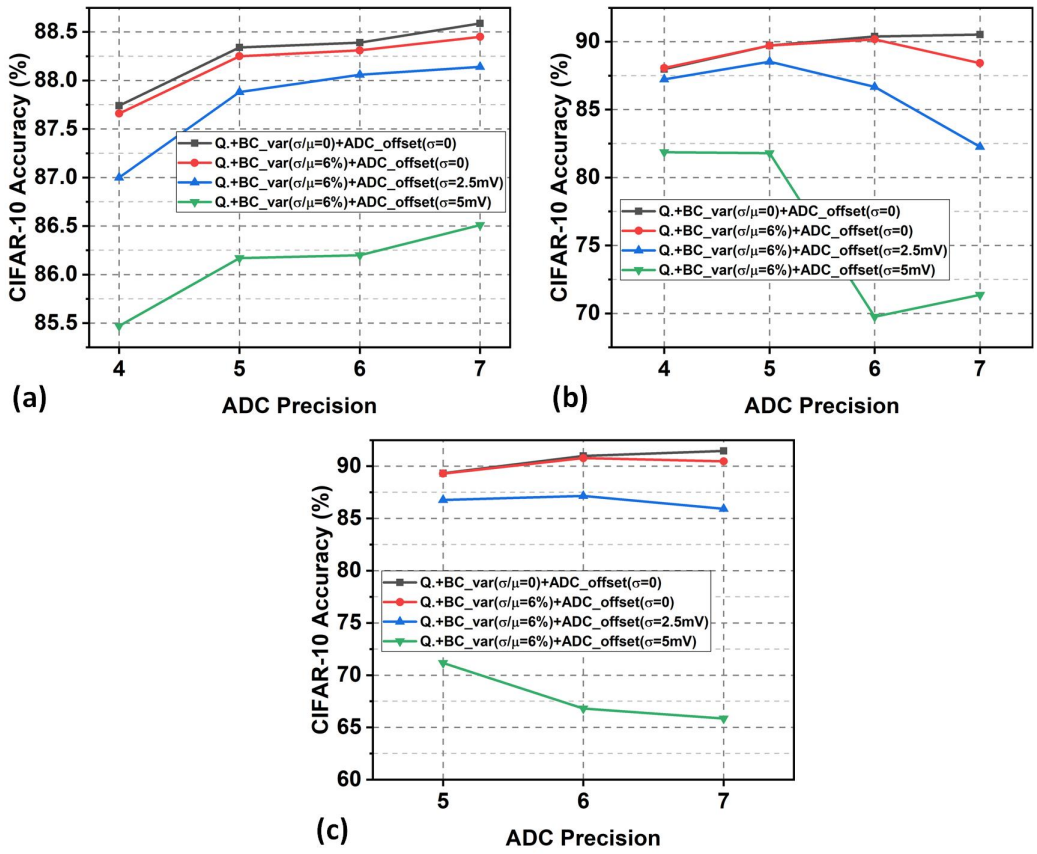



Figure 15. Compilation of Optimal Accuracies at Varying Bit Precisions for ResNet-18 on CIFAR-10

The takeaway from this experiment is that the binary network is more resilient than the 2-bit and 4-bit DNNs. If a low enough noise were to be applied for the 2-bit and 4-bit DNNs, the same trend of increasing accuracy with increasing ADC precision will be observed. We will show more evidence of this in section 3.3.4

### 3.3.4 Experiments of ResNet-18 on ImageNet

Same sets of experiments are also conducted on the ResNet-18 network on ImageNet. Starting with the sweeping across quantization range for a 2-bit weight/activation precision network, in Figure 16. The partial sum distribution for the ResNet-18 on

ImageNet is much different and is much more densely packed around 0. Which is why the applied quantization ranges are much smaller for this network. One key difference between the DNN on CIFAR-10 and on ImageNet is that the DNN on ImageNet is much more sensitive to noise. So a smaller ADC offset noise (1mV) is applied for this set of experiments. Owing to this choice, the 7-bit ADC fairs better than 6-bit and 5-bit ADCs in the noisy cases on ImageNet than it did for the CIFAR-10.

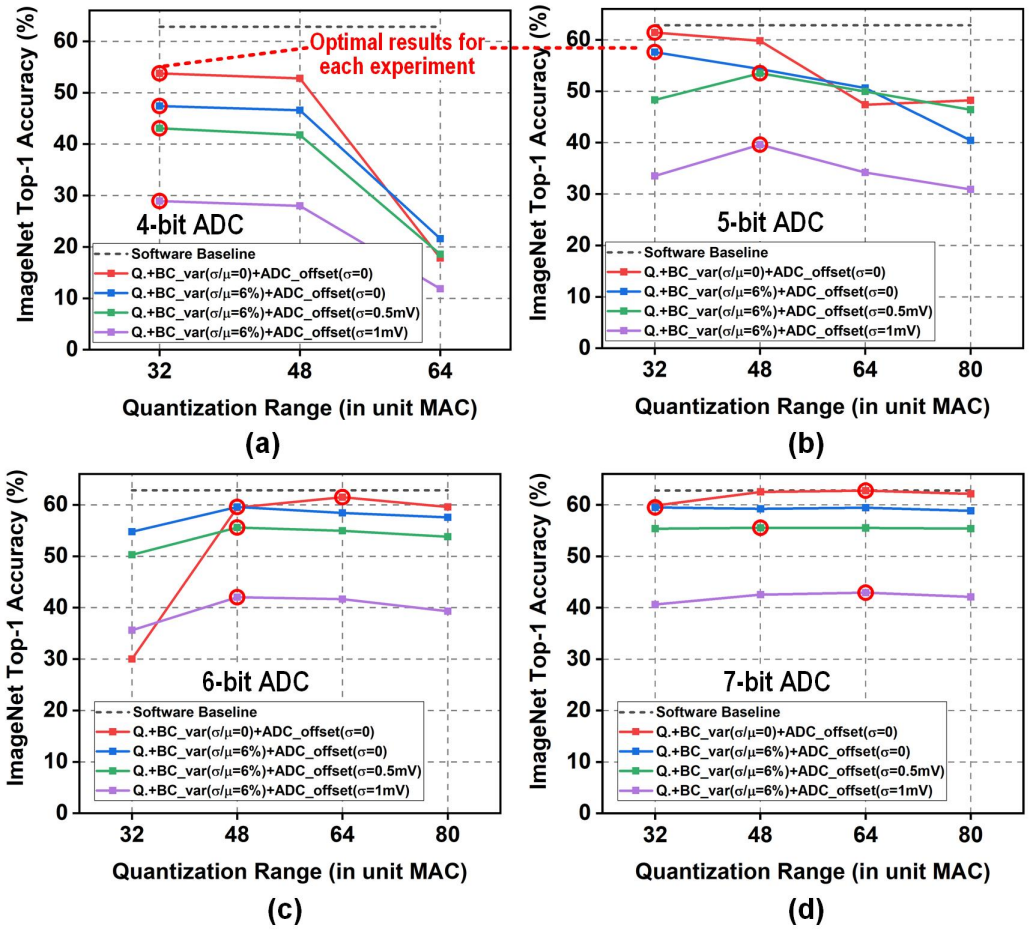



Figure 16. Detailed Simulation Results with the Proposed IMC Modeling Framework for ResNet-18 for ImageNet with 2-bit/2-bit Activation/Weight Precision. For ADC Precisions of (a)4-bit (b)5-bit (c)6-bit (d)7-bit, Results for Various Quantization Ranges, Bitcell Variation and ADC Offset Values.

When compiling the optimal points, as eluded to in section 3.3.3, even for the

noisy cases in the 2-bit ResNet-18 on ImageNet, with increasing ADC precision, the accuracy improves as the noise applied is much smaller (Figure 17)

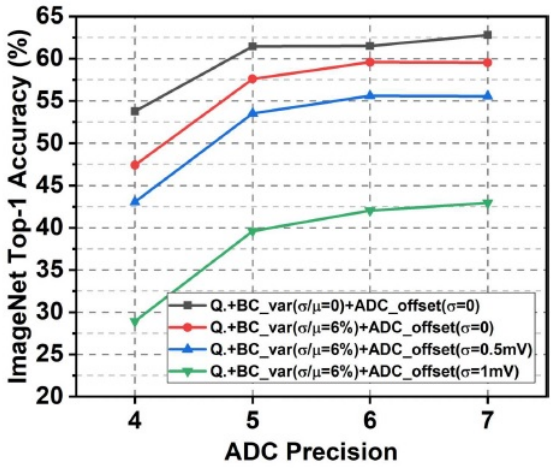



Figure 17. The optimal Results of 2-bit ResNet-18 for ImageNet across Different ADC Precision, Bitcell Variation and ADC Offset, where the Optimal Quantization Range is swept and chosen for each experiment

### 3.3.5 Validation of Model Vs. IMC Chip Accuracy

We proceed to verify the model by running experiments with the on-chip variations observed in the C3SRAM work of Zhang *et al.* (2022). We ran three binary models on CIFAR-10 for the networks, ResNet-18, AlexNet and VGG. The results in Figure 18. The chip based accuracy is observed from 10,000 samples from an ADC output distribution used to calculate the measured error probability. The proposed model closely models the accuracies observed in the C3RAM chip.

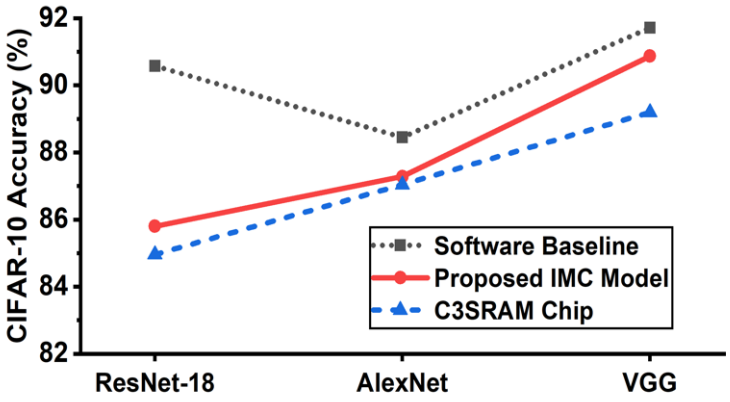



Figure 18. Validating the Proposed Model against the IMC Prototype Chip

## 3.4 Summary

We have presented a simulation framework including different core design parameters (such as the number of active rows, ADC precision and quantization range) to characterize the effect of variation source (such as ADC offset and bitcell variation) on network-level accuracy. Picking an optimal ADC precision however comes down to the noise sensitivity of the model. The proposed IMC modeling framework has been open-sourced and also verified over 3 different DNNs to show a close match to a IMC chip measurement based accuracy. This study will aid as a tool to efficiently implement DNN on IMC hardware while still achieving a high accuracy. The results obtained from the study has been used to optimize the design of a capacitive IMC-SRAM based DNN accelerator chip in Zhang *et al.* (2022).

# Chapter 4

# $K$NN HARDWARE ACCELERATOR USING IN-MEMORY COMPUTING SRAM

## 4.1 Introduction

In this chapter we present our IMC $k$ nearest neighbor (kNN) accelerator. The $k$NN algorithm is a supervised learning tool used for classification in wide range of applications. It can also be used for regression analysis.

In case of its use as classification tool it takes a training data set of vectors representing objects with their classes known as input. For a query object it returns the class membership of the object as output. The class membership of the object is decided based on its distances from its $k$ nearest neighbors in the training dataset. The object is assigned the class to which majority of its $k$ nearest neighbors belong to.

In case of regression analysis, for a query object $k$NN algorithm returns a property value for the object which is the average of the property values of its $k$ nearest neighbors in the training data set.

The algorithm itself is very versatile and it does not make any assumptions about the underlying data. It is widely popular owing to its simplicity and intuitive nature.

There has been a call for more noise aware training approaches to Neural Networks and Machine learning tasks. $k$NN however enjoys a robustness in terms of minimal impact of the noisy data. In addition, it doesn't require a training phase that might require continuous update with the arrival of new data/information.

The algorithm however, suffers from complexity of larger training datasets. Therefore, it is primarily restricted to small and medium sized datasets. And owing to

the online nature of the algorithm it has a high memory usage and consequently the memory access time is very significant to the total run time.

To address this high memory access time we implement the $k$NN accelerator in In-memory Computing. IMC hardware reduces the memory access time by integrating the computing block inside the memory.

## 4.2 The $K$NN Algorithm

To start with, it is necessary to decide on the distance metric that is required to be used for relevant type in the given application. The data objects or data points constitute an abstract object space of an application where the set of attribute or feature values of an object represents the corresponding data object. In mathematical terms such data objects are referred to as vectors. The number of attributes/ features used to represent an object type is the dimension of the corresponding vectors where the i-th attribute/ feature is referred to as the i-th dimensions. For computation of the distance between any two objects is done in terms of the differences in the values of their attributes/features.

### 4.2.1 Distance Metrics

Some of the distance metrics that are found in use are:

- **Manhattan Distance:** It is also known as City block distance or Taxicab distance. Suitable for use in high dimensional data. The Manhattan distance

between two $n$ dimensional vectors (objects) $X$ and $Y$ is computed as:

$$d(X,Y) = \sum_{i=1}^{n} |x_i - y_i|$$

where, $x_i$ and $y_i$ are the values of $X$ and $Y$ in the i-th dimension.

- **Euclidian Distance:** It is the most popularly used straight line distance between two vectors. For high dimensional data its computation becomes costly. Generally, it is preferred for low dimensional data. The distance d(X,Y) is computed as:

$$d(X,Y) = (\sum_{i=1}^{n}((x_i - y_i)^2)^{1/2}),$$

where, $x_i$ and $y_i$ are the values of $X$ and $Y$ in the i-th dimension.

- **Minkowski Distance:** It is a generalized formulation of Manhattan and Euclidian distance metrics given as:

$$d(X,Y) = (\sum_{i=1}^{}((x_i - y_i)^p)^{1/p}),$$

where, $x_i$ and $y_i$ are the values of $X$ and $Y$ in the i-th dimension. Taking $p$=1 and $p$=2 gives us the Manhattan distance and Euclidean distance respectively.

- **Cosine Distance:** In this metric the distance between two vectors is measured as the Cosine of the angle between the two vectors. It is suitable for use in applications where the directions of the vectors are of significance.

$$d(X,Y) = \cos\theta = (X.Y)/(\|X\|.\|Y\|)$$

- **Hamming Distance:** It is defined as the number of bit positions in which two equal length binary strings differ. It is suitable for use in applications where the attribute values are binary.

### 4.2.2 Parameter $k$ Determination:

Proper determination of the parameter $k$ to be used in a given application is critical to correct classification through the $k$NN algorithm. Figure 19 highlights this fact. In the Figure, the red triangle is the query vector to be classified as either dog or a cat. The blue circle shows the nearest neighbors to be accounted for when $k$=3, and the green circle does the same for when $k$=9. In the first scenario, the query is classified as a dog with 2 of 3 nearest neighbors as dog. In the second scenario, the query is to be classified as a cat with 5 of the 9 nearest neighbors being cats.

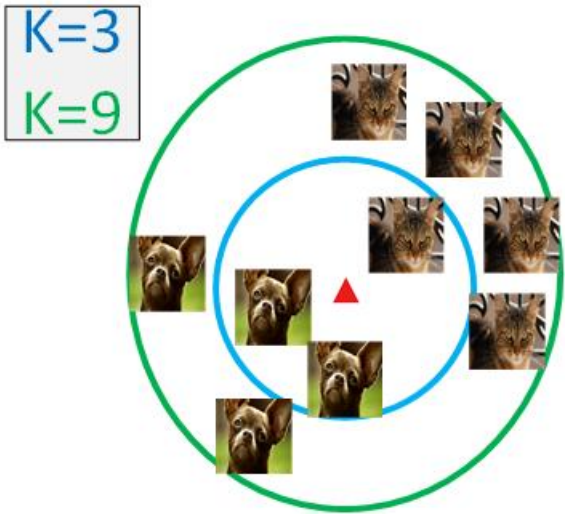



Figure 19. A Demonstration of the $K$NN Algorithm

To achieve the correct selection of $k$, in the first stage of the k-NN algorithm, that can be called the training phase, it takes an annotated training data set from the application domain. The data set consists of records representing individual objects from the application domain with annotations indicating the class to which the individual objects belong. For the given training data set the value for the parameter which produces the best matching result is selected as the $k$ value.

### 4.2.3 Classification

Once the $k$ value is chosen though the training phase, the classification task can be carried out through the following three stages of computations:

1. Distance computation: Compute the distances from all data points in the training dataset to the reference (i.e. the query object).
2. Sorting of the training data set to identify the $k$ nearest neighbors: Use the distance information to sort the data set and pick the $k$-nearest neighbors to the query vector.
3. Majority voting: Count the classes of the $k$-nearest neighbors and pick the class that has the highest count.

## 4.3 Related Works

As eluded to previously, $k$NN algorithm relies on two sets of computation, the distance computation and top k sorting.

Prior works, in Bremler-Barr *et al.* (2015), Imani *et al.* (2017), have used binary and ternary content addressable memories (BCAM/TCAM) as a means to calculate the distance. But since CAMs can provide only a binary result/distance to indicate whether the search vector and the stored vector match or not. Therefore Bremler-Barr *et al.* (2015) use multiple searches to obtain a multi-bit distance, and Imani *et al.* (2017) employ multiple stages of data storage and computation. Neither of these two approaches are conducive to a fast and energy-efficient computation requirement.

### 4.3.1 *K*NN Accelerators

While strictly looking at kNN accelerators, Kaul *et al.* (2016) employed adaptive precision improves latency by reducing the hardware requirement. After computing the current accuracy, only potential winners are kept, eliminating a significant number of vectors. It also supports as its distance parameter both Euclidean and Manhattan distance. Hong *et al.* (2013) performs an approximation of the nearest rather than determining the nearest neighbor. Both of these works use off-the-shelf SRAM, which incur a large amount of memory access and data movements.

Lee *et al.* (2017) used near-data processing to minimize data movement in microns Automata Processor (AP), achieving better $k$NN performance in comparison to other CPUs and GPUs. There have been other works using non-volatile memory based CAMs for $k$NN acceleration. Using MTJ based TCAMs, Imani *et al.* (2017) performed the similarity search for the $k$NN accelerator. Jiang *et al.* (2017); Kaplan *et al.* (2018) employ RRAM for $k$NN acceleration. Choi *et al.* (2018) proposes a novel design that can function as a Binary Neural Network accelerator by morphing from CAMs.

### 4.3.2 The XNOR-SRAM

The XNOR-SRAM developed by Yin *et al.* (2020) computes bitwise XNOR of the query vector and the stored vectors, that can then be accumulated individually for each of the stored vectors discussed in more detail in Chapter 2. As the accumulated result is a bitline XNOR-and-Accumulate (XAC) analog value, it has to pass through an Analog-to-Digital Converter (ADC).

As the XAC of ternary activation and weight (-1/0/+1) is performed over 256

rows, the XAC value ranges between -256 to +256. While ideally all the possible 257 analog values can be mapped to the corresponding digital counterparts. But doing that will incur a significant cost in the form of ADC hardware. It is shown that choosing a non-linear quantization with 11 level can yield a DNN accuracy close to the baseline (with all 257 levels) in Figure 20. So, the output of the XNOR-SRAM is a 11-bit thermometer code that holds the information of similarity between the activation and the weight vectors.

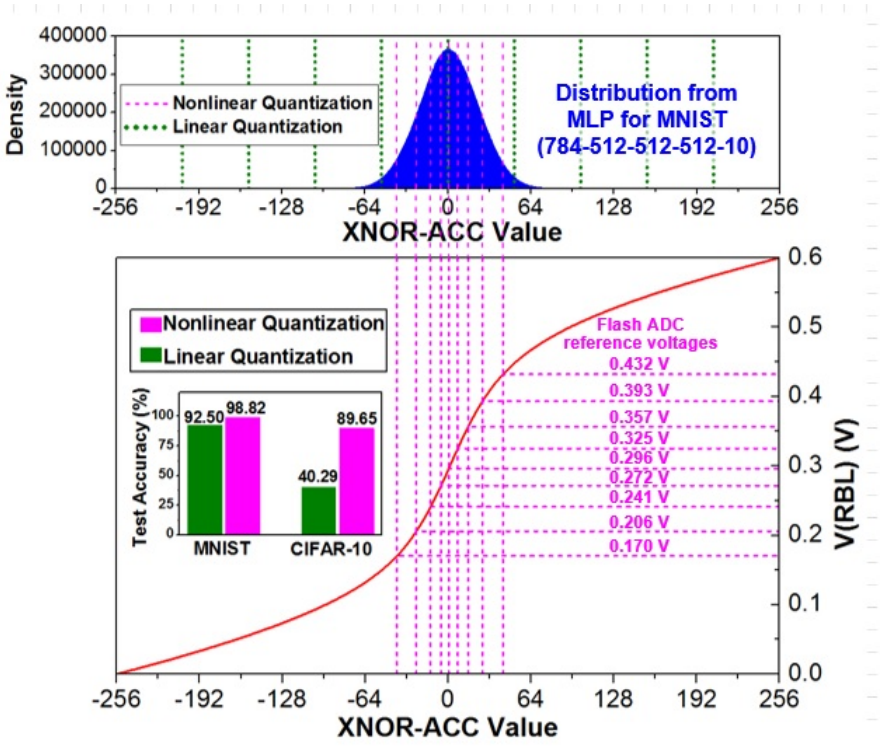



Figure 20. Distribution of the XAC values

**XNOR-SRAM XAC Distance Metric:**

The 11-bit thermometer coded XAC output of the XNOR-SRAM is a measure of the similarity between the query vector and the stored vector in terms of the bit wise comparison between the two vectors. The more the number of bits that match, the

higher is the similarity in the two vectors. The Hamming distance on the other hand gives gives a dissimilarity measure in terms of the number of bits the two vectors differ in. While one gives the similarity measure, the other gives the dissimilarity measure and they are complementary to each other. While using as a distance measure the a lower Hamming distance will imply two closer vectors. On the other hand, a higher XNOR-SRAM XAC value will imply two closer vectors. Therefore, by sorting the XNOR-SRAM XAC output values in a non-ascending order and picking the top-$k$ vectors we can achieve the same purpose as that we achieve by sorting the hamming distance in non-descending order and picking the top-$k$ vectors. The advantage of using this distance metric is the ease of sorting inside the digital sorter and also convenience of easy translation or straight-forward use from the XNOR-SRAM ADC output.

While the XNOR-SRAM work actually encodes the 256-bit XNOR-metric analog distance post-quantization inside ADC, into a 11-bit thermometer code, the example figure 21 illustrates how the XNOR-distance metric for the example 4-bit vectors and its encoding into a 11-bit thermometer code.

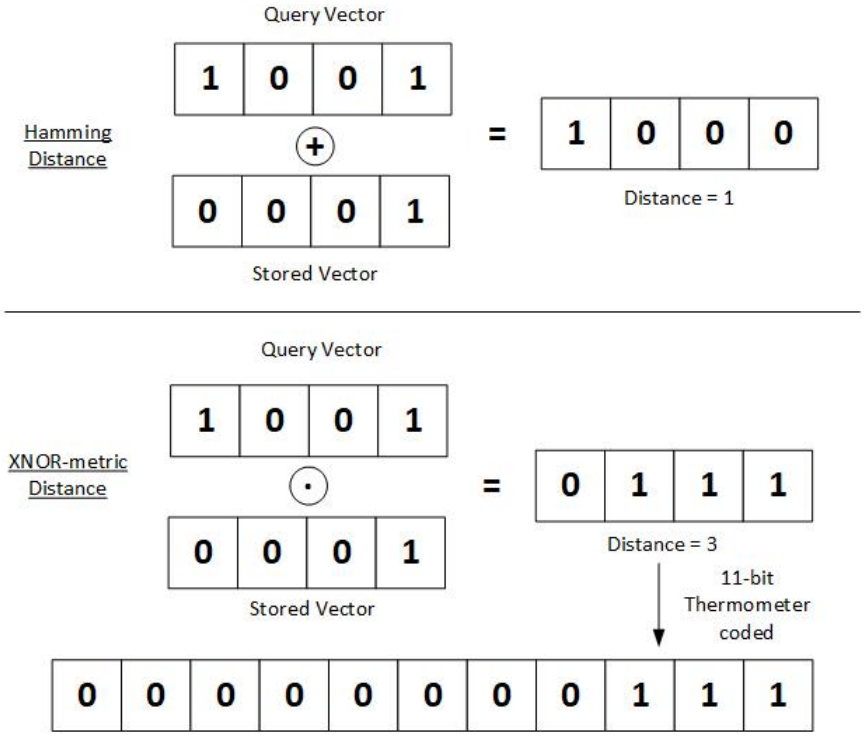



Figure 21. A Demonstration of the Complementary Nature of the Hamming Distance and XNOR-SRAM XAC Distance

## 4.4 The Proposed Work

The output from the XNOR-SRAM is the 11-bit thermometer code which holds the information of the similarity between the Activation and Weight Vectors. The XNOR-SRAM acting as the distance computation engine, needs to be integrated with a compatible digital top-$k$ sorter, Figure 22. The figure also details how the XNOR-SRAM can function as a BCAM/TCAM which is beyond the scope of discussion of this thesis work. Before diving into the digital sorter architecture, more detail on this XNOR-based distance metric is provided.

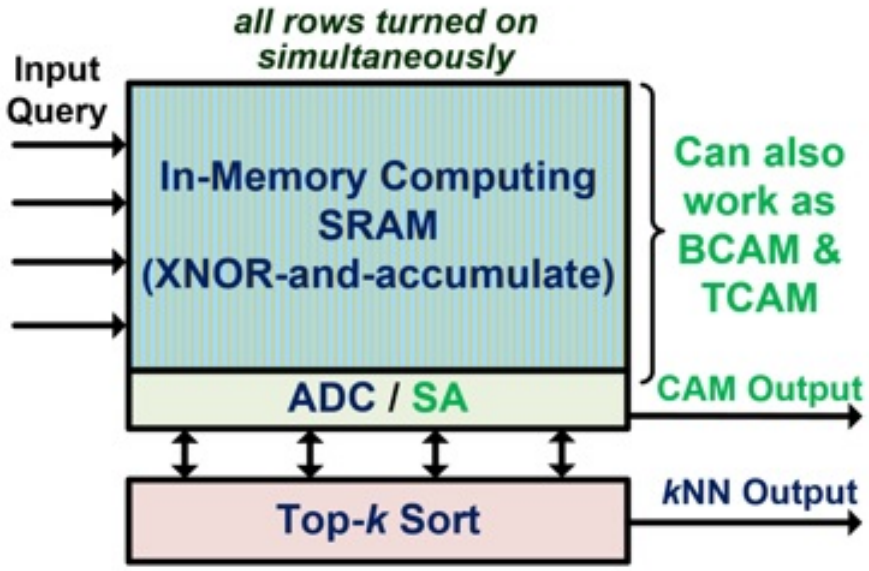



Figure 22. Proposed kNN accelerator based on In-memory Computing SRAM

### 4.4.1 The Digital Sorter

The XNOR-SRAM operates column-by-column meaning that one set of ADC outputs are sent over to the digital sorter every clock cycle, finishing at the the end of the 64th clock cycles. The digital sorter then sorts the k-top neighbors.

Figure 23 shows the digital sorter module architecture. The input to the digital sorter is a 11-bit thermometer code from the XNOR-SRAM functioning as distance computing engine.

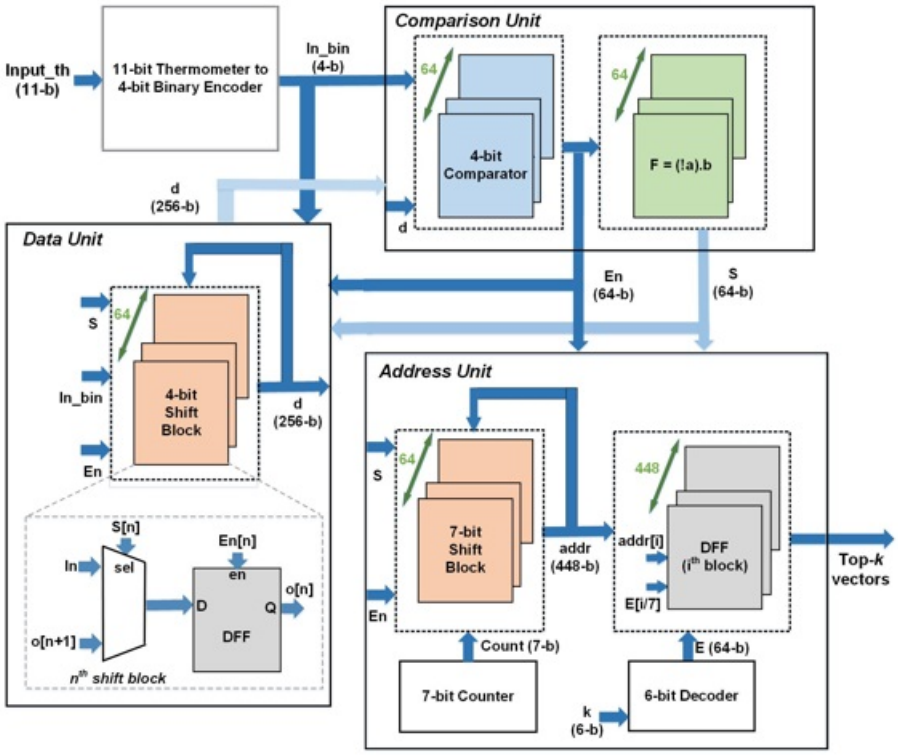



Figure 23. Block Diagram of the Digital Sorter Computational Module

The sorter architecture employs an encoder to compress the 11-bit input distance information into 4-bit and saves energy. The comparison unit generates the Shift and Enable signals that will be used to control the data and address blocks.

The data unit maintains a sorted list of distances. When a new distance vector arrives in the data unit, this vector is compared to the all the vectors in the sorted list. Any vector smaller is shifted towards the Least-Significant end of the sorted list. And an appropriate spot in the list is determined to place the incoming distance vector.

In parallel, a similar list is maintained in the Address Unit storing the address of the stored value in the memory array to which the distance value corresponds to. The $k$ nearest neighbors, can then be picked by fetching the stored vectors in the memory at the top $k$ addresses.

To give a more detailed recounting of the events. To start off, in the data unit, all

the shift registers are set at 0. Once the new ADC output comes in and is encoded into the 4-bit value, it is compared with the sorted list, to determine its position in the list. The smaller-value vectors are down moved down the sorted list by one spot to make room for the new arrival.

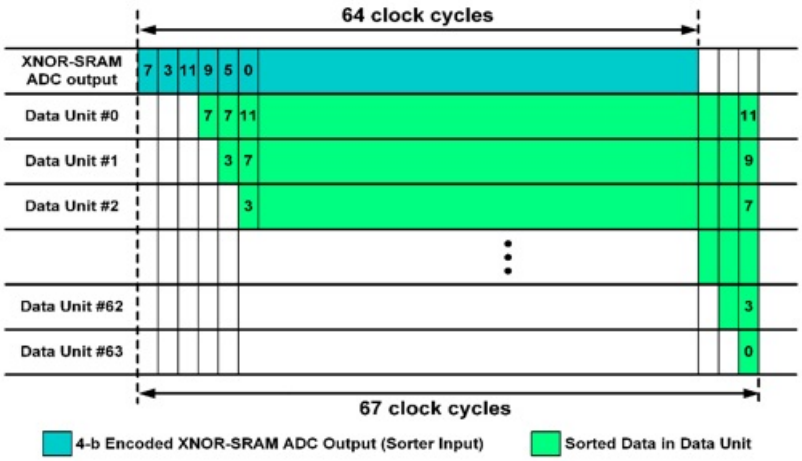



Figure 24. Timing Diagram and Operation of the $K$NN Accelerator

The timing diagram for the full operation involving all of the 64 columns is shown in Figure 24. The first data and address is introduced in the data and address unit sorted lists starting in the 3rd cycle and every subsequent cycle. After going through all of the columns, the sorted list is ready in the 67th cycles, at which point the top-$k$ values are sent out. The algorithm implementation has an O(n) complexity in the number of clock cycles.

### 4.4.2 Submodules Description:

The Comparison unit generates the Shift 'S' and Enable 'EN' signal. The 64b Enable 'EN' signal is obtained by using 64 sets of 4b comparators to compare the query vector and the store vectors. The comparators return a 1 for a values less than,

and a 0 for a value greater than or equal to the query vector. The 64b shift 'S' signal is obtained corresponding to the position of first '1' in 'En'.

Inside the data unit, we have a set of 64 4b shift registers. Each of these shift registers are composed of a multiplexer and a D-FF. The multiplexer selects to feed in either the query vector or the next closest stored value from the sorted list, using the 'S' signal. The output of this multiplexer is then either passed through the shift register if 'En' is 1.

The address unit shares the 'S' and 'EN' signal to continually maintain a sorted list corresponding to the data unit list. The $k$ value is also fed into the address unit to output the addresses of the top-$k$ vectors

### 4.4.3 Scaling to Larger Vector Sizes

One XNOR-SRAM array is composed of 64 columns. To scale up the digital sorter module to 128 and 256, multiple instances (2 and 4 respectively) of the XNOR-SRAM array are used.Also, the number of shift registers inside the data unit and address unit scales at the same rate since, the size of these units holds a relationship with the vector size.

## 4.5 Experimental Results

Both of the primary modules in the proposed $k$NN accelerator design, the XNOR-SRAM array and the digital top-$k$ sorter module, were implemented in 65nm CMOS tech. The XNOR-SRAM has been fabricated in a prototype chip, the obtained measurement results were reported in Yin *et al.* (2020). The digital sort module power

analysis simulations and post-APR (automated placement and routing) static timing analysis were performed in Synopsys Primetime. Additionally, RC parasitic annotated netlists and node toggling activity were used to obtain the latency and power results.

In Figure 25, the digital sort physical design is placed alongside the die photo of the XNOR SRAM array. On the left, we have the 256x64 design and on the right, a scaled version of the design the 256x256 design is shared.

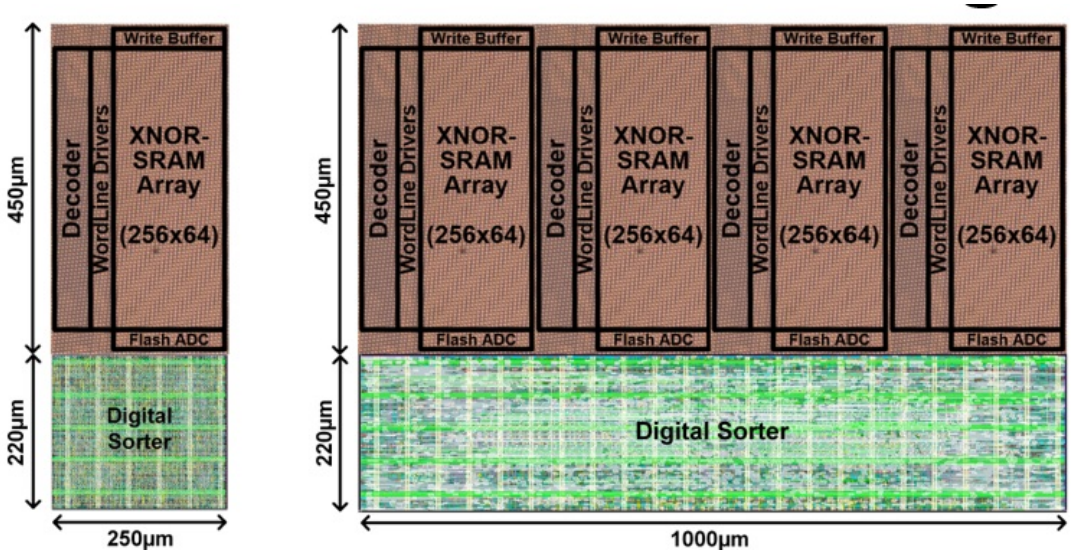



Figure 25. Physical Design Diagram of the Proposed $K$NN Accelerator. The Accelerators Supporting (A) 64-Vector, (B) 256-Vector Operations.

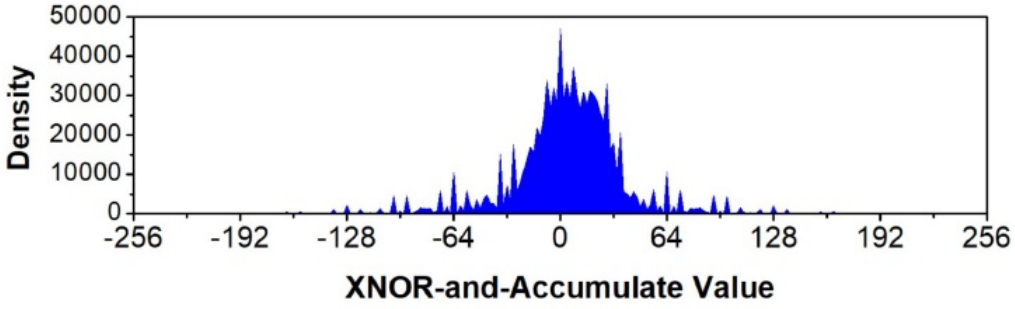



Figure 26. Random Data Distribution of XNOR-and-Accumulate Values for Each Column

For the experiments, the distribution of the $k$NN workload used for both the query vector and the stored vectors is shared in Figure 26. The distribution is corresponding to 1 million pairs of 256-bit random data. Using the reported power corresponding to each of the XNOR-and-accumulate value reported in Yin *et al.* (2020), the average power is calculated. Since the distribution is heavily distributed around zero, the power consumed is very similar to when the XAC value is zero (4.4mW @ 1.2GHz).

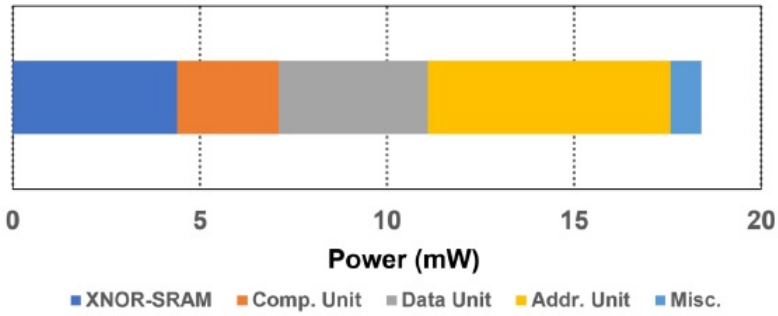



Figure 27. Power Breakdown of the $K$NN Accelerator for Vector Size of 64

The power breakdown of each of the submodules is shared in Figure 27. The XNOR-SRAM boasts a very high power-efficiency and shows a similar power to the primary submodules in the digital sorter design. The comparison unit takes up less than (1/5)th of the power share, while the data unit takes up just over (1/5)th of the total share. The address unit handles the more bit consuming addresses (in this design), and thus takes up about (2/5)th of the power share.

In Figure 28, we present the throughput and power of the multiple digital sorter designs. As we scale up from a vector size of 64 to 128 and 256, the complexity of the sort module increases and also has longer interconnects resulting in a lower maximum operating frequency. The operating frequency for the designs with vector sizes 64,18 and 2565 were observed to be 3GHz, 1.2GHz and 0.6GHz respectively. To compare the designs the power consumption is characterized at two frequency nodes, at 1.2GHz

for vector sizes 64 and 128 and at 0.6GHz for all three designs. A roughly linear power curve is observed at both of the nodes. Additionally the throughput drops by more than half when scaling the design by 2x. This is owing to the decrease in maximum operational frequency and the increased latency.

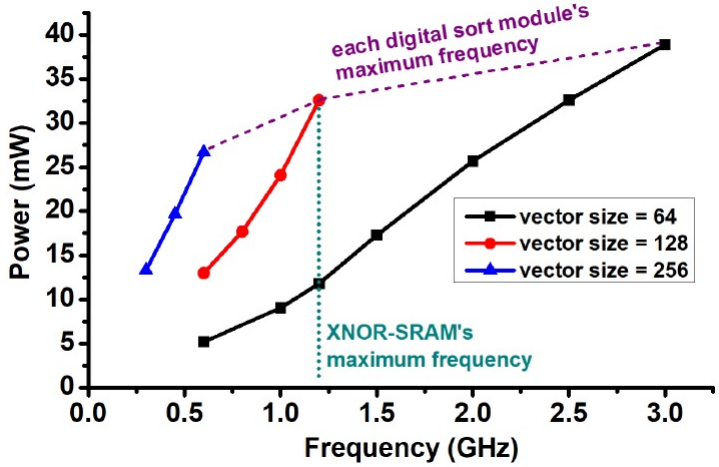



Figure 28. Frequency and Power Consumption of the Digital Sort Module for 64, 128 and 256 Vector Sizes

We also care about the Energy-per-query shared in Figure 29. For vector size 64,the 17.9million queries per second is achieved, consuming 16.2mW at 1.2GHz. The energy-per-query increases at a linear rate when scaling the vector size.

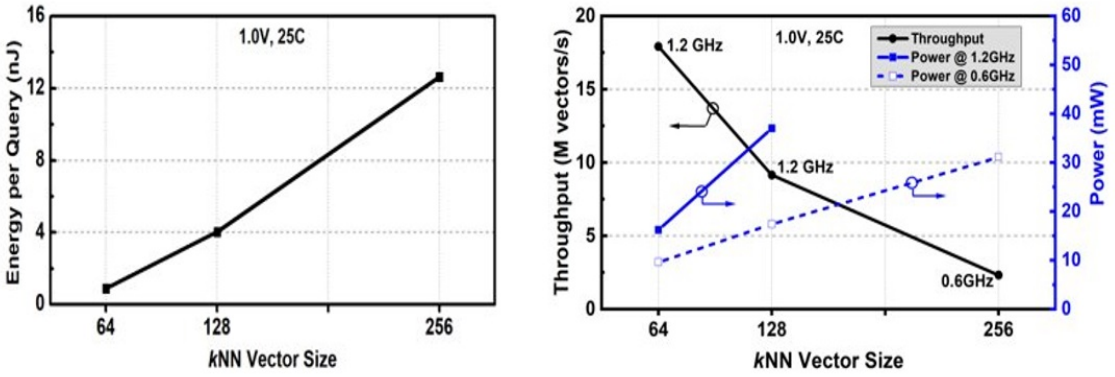



Figure 29. (Left) Energy Consumption of the Proposed $K$NN Accelerator across Different Vector Sizes. (Right) Throughput and Power Consumption of the Proposed $K$NN Accelerator, including both XNOR-SRAM and Digital Top-$K$ Sorter.

## 4.6 Comparison with Related Work

Moving on to a side-by-side comparison of our kNN accelerator with other NN works in Table I. Our design supports 3 array sizes: 256x64, 256x128, 256x256. As the distance information is readily available in the memory, the XNOR-SRAM, the area saved is significant. 1.94x smaller than the next closest design. The throughput falls short of that shown by the Intel work but has much less power consumption. Provides much better throughput than the remaining works. And as can be expected form these stats, the energy/query vector shows a >4.8X improvement.for vector size 128.

TABLE I
*k*NN ACCELERATOR COMPARISON

| | This work | Kaul *et.al* | Hong *et.al* | Kim *et.al* |
|---|---|---|---|---|
| CMOS Tech. | 65nm | 14nm | 130nm | 130nm |
| Vector Dimension | 256x64 / 256x128 | 128x128 | 128x128 | 272x128 |
| Area ($mm^2$) | 0.17 / 0.34 | 0.33 | 2.26 | 1.44 |
| Supply (V) | 1.0 | 0.85 | 1.2 | 1.2 |
| Frequency (GHz) | 1.2 | 0.42 | 0.2 | 0.2 |
| Throughput (vectors/s) | 17.9M / 9.15M | 21.5M | 0.13M | 0.06M |
| Power (mW) | 18.4 / 37.0 | 73 | 65 | 44 |
| Normalized Energy/Query[1] | 0.89 / 4.03 | 19.6 | 170 | 250 |

[1]Energy is normalized to 65nm, assuming capacitance scales by 0.7 in each technology generation, and used supply voltage values in the table.

## 4.7 Summary

To sum it up, *k*NN hardwares require a lot of memory access which is where in-memory can provide a massive boost in saving time and energy-efficiency. The XNOR SRAM offers BCAM and TCAM schemes with binary stored vectors. The Accelerator supports three vector sizes 64,128 and 256. The design implementation in 65nm CMOS can perform at up to 17.9 M query vectors/s while consuming 18.4 mW at 1.2GHz . This work also demonstrates an energy per query vector improvement of over 4.8x compared to prior works.

Chapter 5

# FP-IMC: A 28NM ALL-DIGITAL CONFIGURABLE FLOATING-POINT PRECISION IN-MEMORY COMPUTING MACRO DESIGN FOR HIGH-ACCURACY DNNS

## 5.1 Introduction

As discussed in Chapter 2, every layer of a DNN requires performing of a large number of multiplication and accumulation (MAC) operations. These consume a large amount of time and energy in traditional hardware. Requirements of high precision and accuracy needs the MAC operations to be performed in floating point (FP) format which adds significantly to the cost. Having IMC hardware accelerators can reduce these costs very significantly. Embedding these accelerators in edge devices require special care to bring the costs in terms of energy efficiency, space efficiency within the acceptable limits of these devices. To support real-time applications achieving the desired speed of processing becomes crucial. In this chapter we present the design and 28nm implementation of an all-digital IMC accelerator macro for a high accuracy DNN plane block with configurable floating-point precision MAC operations.

State-of-the-art works have demonstrated implementing DNN algorithms to achieve high accuracy for a wide range of applications such as computer vision, and autonomous driving. But these works are very computation intensive and demand a large memory. Floating Point (FP) precision is necessary during training exhibiting high-accuracy down to 8-bit FP (FP8) (Wang *et al.* (2018)). While most DNN inference workloads

use fixed-point precision, Jouppi *et al.* (2021) shows that using FP precision can also aid in avoiding accuracy loss in complex inference tasks.

Most of the IMC macros in the literature have concentrated on fixed-point precision operations (Chih *et al.* (2021),Zhang *et al.* (2022)). Tu *et al.* (2022) reports a FP computation using near-memory computation, but lacks in tighter integration of computation and memory. A floating point IMC was reported in Jeong *et al.* (2022), but additional memory was dedicated to the mantissa product and exponent sum with a 64% area overhead. Wu *et al.* (2023); Guo *et al.* (2023) report floating-point IMC macros at 16-bit brain FP precision. Wu *et al.* (2023) faces accuracy degradation with the use of hybrid analog circuits, while Guo *et al.* (2023) suffer due to approximate computing/quantization.

A novel floating-point precision IMC (FP-IMC) macro is presented here to immerse the floating-point computation into the weight memory. This work supports two FP8 configurations:

1. Normal 8-bit floating-point (FP8) precision adopting the (1,5,2) notation, which is 1-bit sign, 5-bit exponent and 2-bit mantissa.
2. Block 8-bit floating-point (BF8) precision adopting the (1,2) notation. Here a common 5-bit exponent is assumed for all words, and implied in the notation.

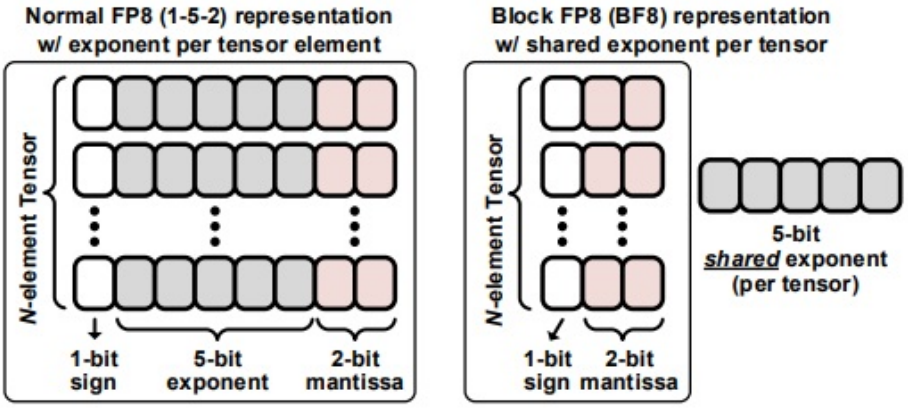



Figure 30. Illustration of the FP8 and BF8 Schemes

Figure 30 shows the two floating point precision implemented in the architecture. The architecture was implemented on a 28nm CMOS prototype chip, achieving a normalized throughput of 166.9/606 GFLOPS/kb and energy-efficiency of 12.1/66.6 TFLOPS/W for FP8 and BF8 precision modes respectively showing a large improvement over state-of-the art floating point IMCs.

## 5.2 FP-IMC Architecture and Operation

The overall architecture of the proposed FP-IMC macro is shown in Figure 31. A 64x64 (4kB ) macro is implemented, which is divided into 8 columns of 64x8 sub-macro arrays. The 512-bit input is fed to each these sub-macros in parallel to output the FP8 MAC output.The sub-macros hold 64 rows of 8-bit weights that are accumulated towards one 8-bit FP8 output. Each of these rows has 8 SRAM bitcells to store one word of FP8 weight. FP multiplications are performed inside these SRAM bitcells when the 8-bit inputs are streamed into them, and this multiplied output goes through the mantissa,exponent handling sub-modules and the normalization modules before being accumulated inside the adder trees.

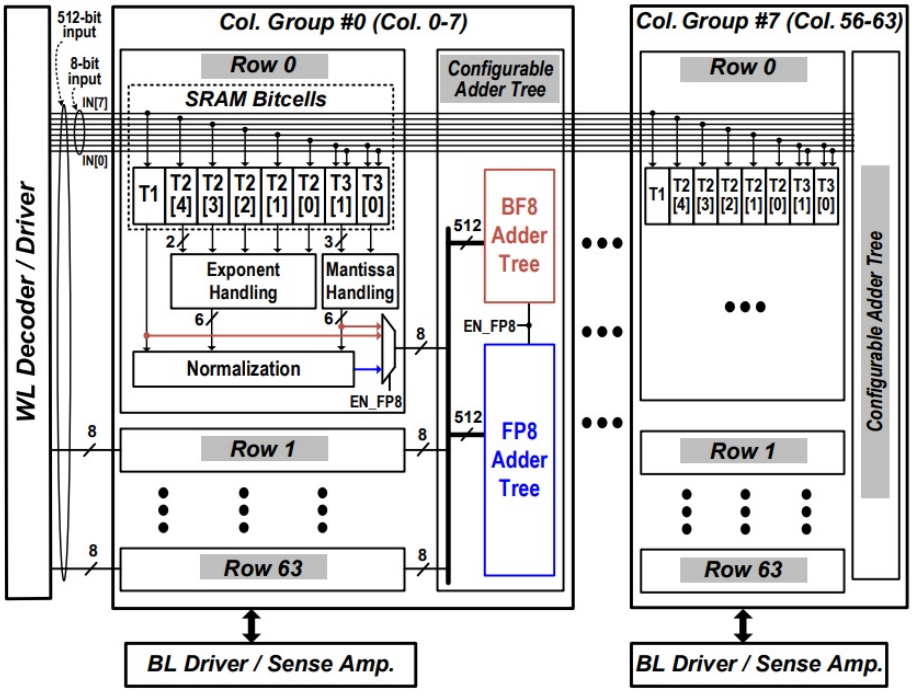



Figure 31. Proposed FP-IMC Macro and Architecture Design

### 5.2.1 FP Multiply Operation

The FP8 multiplication of the input (IN) and the weight (W) goes as:

$$\begin{aligned} IN \times W &= (-1)^{IN[7]} \times e^{IN[6:2]} \times 1.(IN[1:0]) \\ &\quad \times (-1)^{W[7]} \times e^{W[6:2]} \times 1.(W[1:0]), \end{aligned} \tag{5.1}$$

$$\begin{aligned} &= (-1)^{(IN[7] \oplus W[7])} \times e^{(IN[6:2]+W[6:2])} \\ &\quad \times 1.(IN[1:0]) \times 1.(W[1:0]), \end{aligned} \tag{5.2}$$

It is challenging to implement the FP multiplication operation owing to its complexity. In our approach, we breakdown this multiply operation into three less complex sub-operations:

- A XOR operation for the sign bit (bits [7]).

- A 5-bit addition for the exponent bits (bits [6:2]).
- A 2-bit multiplication for the mantissa bits (bits [1:0]).

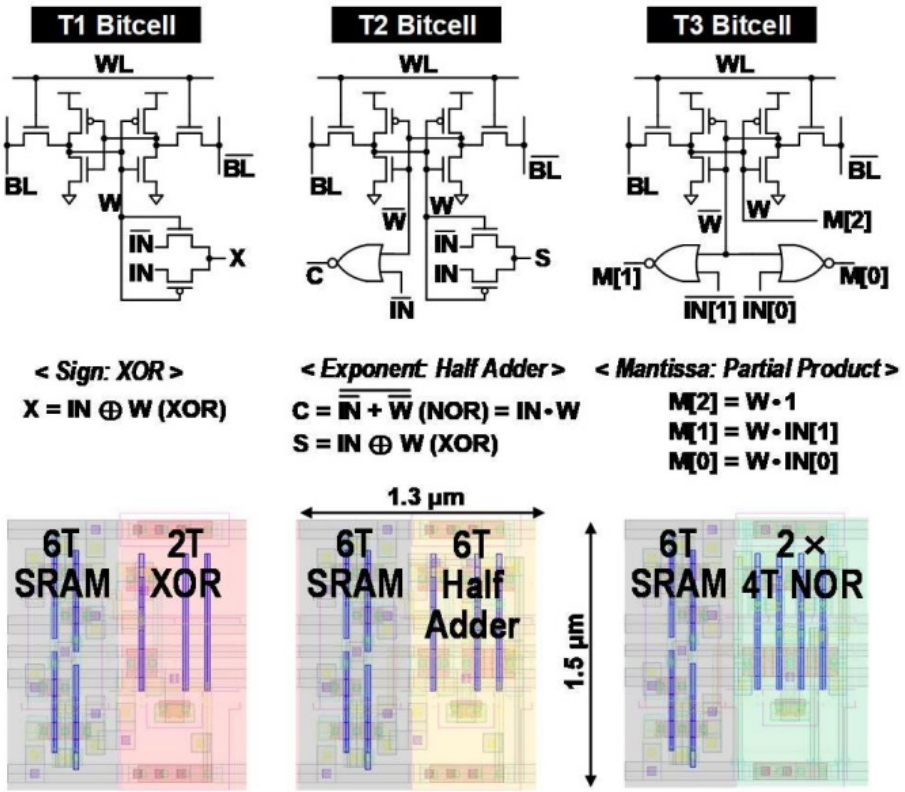



Figure 32. T1/T2/T3 Bitcell Designs, Compute Functionality and Layout

Three bitcells (shown in Figure 32,top), each dedicated to a sub-operation, are designed to store the 8-bit weights and perform the multiply operation efficiently. Each of the bitcells has the 6-T SRAM and some additional transistors to support the compute in-memory. **T1** bitcell performs a bitwise XOR for the input and weight sign bit. **T2** bitcell performs a sum and a carry operation for the input and weight exponent bits.The sum operation is carried out as a XOR function with the use of 2T pass gate, and the carry function with a 4T NOR gate. Finally, the **T3** bitcell uses two NOR gates to obtain the 3-bit partial product, $M[2:0] = (1'b1, I[1:0] \times W_M)$.

The physical design layouts for each of the three bitcells are shown in Figure (32,bottom). Each of them are regularized to an area of 1.3x1.5µm$^2$.

The **T1**/**T2**/**T3** bitcells perform the bit-wise multiplication of the input and weight bits. Additional supporting modules are necessary to integrate the bitcell outputs into a cohesive FP8 product. Figure 33 shares the supporting logic designed for this ensuing computation.

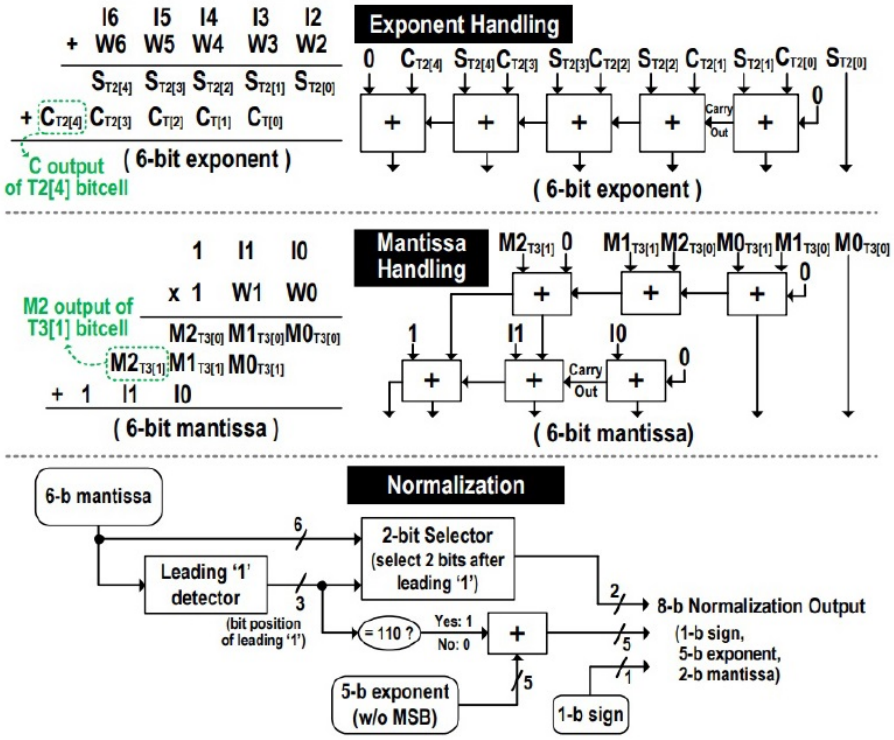



Figure 33. Block Diagram of Exponent Handling Module (Top), Mantissa Handling Module (Middle), and Normalization Module (Bottom)

In the **exponent handling** module (Figure 33,top), the five **T2** bitcells sum and carry outputs are accumulated into 5-bit exponent sum of the input and weights using an energy-efficient ripple carry adder. Since the exponents of the weight and sum are common and known in for BF8 mode, the **exponent handling** module is turned

off during the BF8 mode. In the **mantissa handling** module (Figure 33,middle), the sets of 3-bit partial products from the two **T3** bitcells are summed up to obtain the 3-bit mantissa multiplication. The $3^{rd}$ mantissa bit is the hidden 1 associated with the FP notation. Before, these products can be accumulated they have to be normalized to the FP8 notation, for the FP8 mode inside the **normalization** module (Figure 33,bottom). However for the BF8 mode, since the exponent of the weights and inputs are common, the adder tree is a fixed-point one and does not require a normalization (Will be covered in more detail in section 5.2.2). The **normalization** module detects the leading '1' in the mantissa product and adjusts the mantissa and exponent accordingly to adhere to the adopted FP8 notation.

As was evident from the detailed description of the sub-modules, some of them can be turned off during the BF8 mode. This is done by using a multiplexer to clock-gate the **mantissa handling** and **normalization** modules to be turned off during BF8 mode reducing the power consumption. The 'En_FP8' signal is used to toggle between the two modes inside the architecture.

### 5.2.2 FP Accumulate Operation

The MAC output of a column group can be expressed as:

$$Col_{OUT} = (IN_0 \times W_0) + ... + (IN_{63} \times W_{63}) \tag{5.3}$$

The accumulation of the 64 rows of partial products inside each column group is done using a configurable adder tree. The adder tree can be configured to either the BF8 or FP8 mode using the 'En_FP8' signal. Based on this signal, either of the two sub-adder trees are activated. A generic adder tree to support both the modes is viable but suffers from a very high power consumption. To address this, dedicated

adder trees are designed for either mode which can be turned on/off to lower power consumption.

The BF8 adder tree closely resembles a traditional fixed-point adder tree with no normalization modules necessary after each step. The common exponent is then accounted for at the end of the adder tree once all the accumulations are done to finally, normalize the output to the BF8 notation.

Inside the FP8 adder tree however, after each set of FP addition, the sum has to be normalized to the FP8 notation. This incurs a significant area overhead in comparison to the BF8 adder tree. The sum is performed in 16-bit FP precision, to avoid overflow loss. For the 64 rows, the first level of the adder tree contains 32 16-bit FP adders, which drops to 16 16-bit FP adders in the next level and so on. The output of the final level passes through the FP8 normalizer module to format the output to the FP8 notation.

The sub-modules active during the BF8/FP8 mode and the corresponding timing diagrams are shown in Figure 34. When the 'En_FP8' signal is set to 1(0), the FP8 (BF8) mode is turned on. and the BF8 (FP8) adder tree is clock-gated (Figure 34, left). Additionally, during the BF8 mode the **T2** bitcell, the **exponent handling** module and the **normalization** modules are turned off as well. The simple BF8 mode logic results in a low latency and the BF8 output can be obtained in a single cycle. The FP8 logic, however, is much more complex and has a significantly larger latency. To lower the FP8 mode latency, we employ a 3-stage pipeline to reduce the clock-cycle period significantly. The serially fed FP8 inputs can be computed parallely without any stall, to output one FP8 MAC value every cycle resulting in a higher throughput. The 64x64 macro array can perform 64x2x8 =1024 FP operations per cycle for each mode.

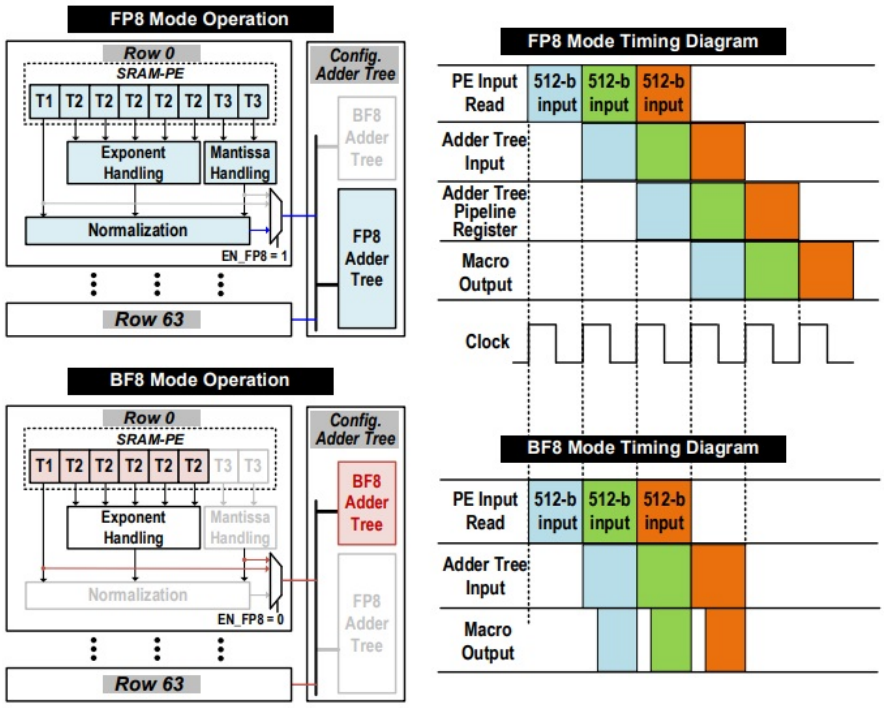



Figure 34. (Left) Active Sub-modules in FP8 and BF8 Mode. (Right) Timing Diagrams for FP8 and BF8 Modes

## 5.3 Chip Measurements and Results

The prototype chip was implemented on 28nm CMOS, consuming an area of 0.71 $mm^2$ (Figure 35). Dynamic voltage scaling experiments were run on the chip, the results of which are illustrated in Figure 36.

For both the modes the chips were operational down to 0.56 V supply at room temperature. A pseudo-random linear-Feedback Shift Register was used to feed in inputs continually to properly gauge the power measurements. For weights, non-zero weights were randomly assigned within the minima and maxima of the FP8 range.

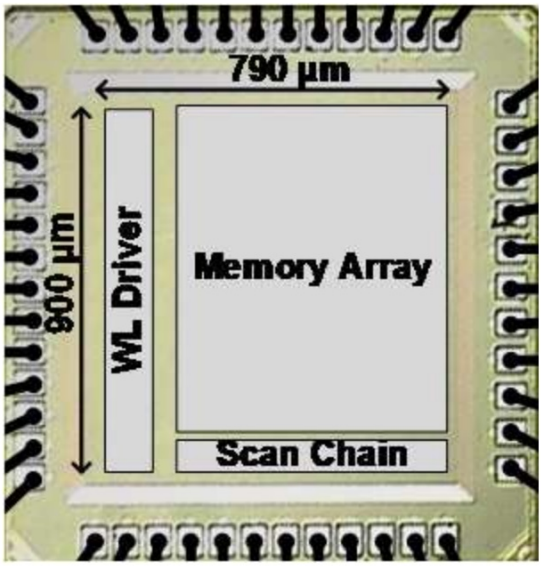



Figure 35. FP-IMC Chip Micrograph

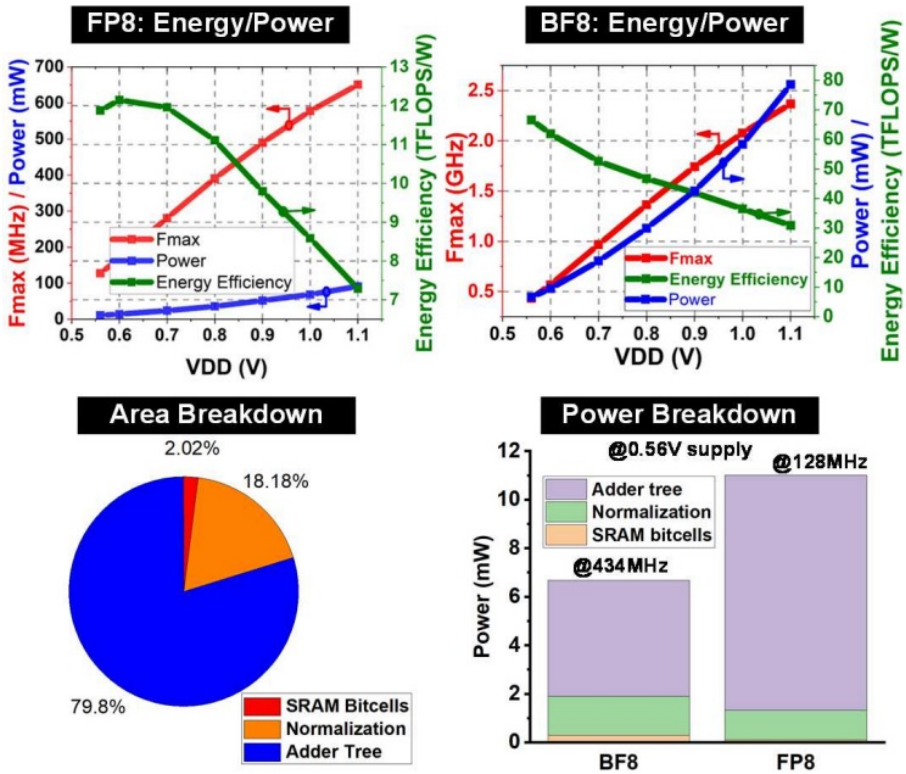



Figure 36. (Top) Energy-efficiency/ Power of FP8 and BF8 Scheme with Voltage Scaling (Bottom) Area and Power Breakdown

The highest energy-efficiency were noted to be:

- For FP8, 12.1 TFLOPS/W at 0.6 V.
- For BF8, 66.6 TFLOPS/W at 0.56 V.

The highest throughput were noted at 1.1 V to be:

- For FP8, 0.67 TFLOPS at 650 MHz.
- For BF8, 2.42 TFLOPS at 2.4 GHz.

The Figure 36, bottom shows the power and area breakdown of the FP-IMC Macro. The power breakdown was made for both modes at 0.56 V. while the total power was calculated from on-chip measurements the module level breakdown is from post-layout simulation. Also, since the **normalization** module is turned off during BF8 mode, the multiplexer power and area are listed as '**normalization** module' in this breakdown. For FP8, mode the Adder tree incurs a significant power and area cost owing to the Floating-point accumulation. The BF8 adder tree with its fixed point adders and in general handling of less bits incurs a much smaller portion of the total power and area cost. That being said, the overall macro makes a major stride towards an energy-efficient solution.

## 5.4 Comparison to Other Works

To illustrate the energy-efficiency benefits of the IMC, we also implement the a conventional (non-IMC) 8-bit MAC computation array of the same size 64x64, with off-the-shelf SRAM array generated by commercial memory compiler in 28nm CMOS. Comparing our work to this non-IMC array implementation performing the same MAC functionality, our macro achieves 2.8 $\times$ higher energy-efficiency.

Table II shows the comparison of this work with other related works. Our work shows the highest macro-level throughput normalized per IMC array size. This is owing to the pipelining and the high operating frequency achieved in both modes, while some works targeted system level integration. Compared to the FP8 mode in Jeong *et al.* (2022), the FP-IMC achieves 7.4x higher energy-efficiency. BF8 mode offers the highest energy-efficiency, which offers at least 2.1 × better energy-efficiency than the next best.

TABLE II
COMPARISON WITH PRIOR FLOATING POINT IMC WORKS

| | VLSI 2021 [5] | ISSCC 2022 [6] | ESSCIRC 2022 [7] | This work |
|---|---|---|---|---|
| **Technology** | 28nm | 28nm | 28nm | 28nm |
| **Memory** | 64Kb | 96Kb | 16Kb | 4Kb |
| **CIM Bitcell** | 6T | 10T | 8T/10T | 8T/12T |
| **Voltage** | 0.68-1.1V | 0.6-1.0V | 0.5-0.9V | 0.55-1.2V |
| **Precision** | BF16 | BF16 INT8-32 | FP8-FP32 | FP8/BF8 |
| **Frequency** | 250 MHz | 50-220 MHz | 53-403 MHz | 650MHz /2.4GHz |
| **Area ($mm^2$)** | 1.62x3.6 | 3.28x2.04 | 1.0x1.0 | 1.2x1.3 |
| **Normalized[1] Peak Throughput (GFLOPS/Kb)** | 1.87 (1.1V, BF16) | 11.3/14.1 (1.0V, BF16/INT8) | 3.62 (0.9V, FP8) | 166.9/606.0 (1.1V, FP8/BF8) |
| **Energy-Efficiency (TFLOPS/W or TOPS/W)** | 1.43 (BF16) | 29.2/36.5 (0.65V, BF16/INT8) | 1.64 (0.5V, FP8) | 12.1/66.6 (0.55V, FP8/BF8) |

[1]Throughput normalized per Kb of CIM memory size.

## 5.5 Summary

This work was focused on addressing the key challenge with implementing the MAC operations that are crucial to a DNN in an IMC array. We presented a novel solution proposing a floating-point precision IMC macro where the float-pointing

computations are immersed into the weight memory storage. The proposed FP-IMC supports two FP8 configurations of (1) normal 8-bit floating-point (FP8) precision with 1-bit sign, 5-bit exponent, and 2-bit mantissa (1-5-2), and (2) 8-bit block floating point (BF8) precision with a shared exponent among 64-element weight tensor . Our proposed FP-IMC macro was implemented in a prototype chip in 28nm CMOS, and achieves 12.1 TFLOPS/W for FP8 and 66.6 TFLOPS/W for BF8.

Chapter 6

# RFP-IMC: A 28NM RE-CONFIGURABLE FLOATING POINT IMC MACRO SUPPORTING PRECISIONS FP8 TO FP32.

## 6.1 Introduction

In the FP-IMC work, presented in Chapter 5, we primarily focused on a novel solution to floating point MAC operations. The mini float (FP8 and BF8) precision incorporated in that work offers a trade-off of a small area overhead for higher accuracy. But for some real-life practical applications, for instance Image Recognition in Autonomous Vehicles, very high accuracy is required. This calls for some elegant solution to implement high precision FP MAC models along with high energy-efficiency.

In this work, we strive to improve the energy-efficiency over the previously presented architecture while simultaneously incorporating higher precision options. We incorporate options of 7 FP mode with precisions starting from 8-bits up to 32-bits. We provide two solutions for implementing the computation. The first one is an economical solution with lesser hardware requirement that performs operations on a block of inputs with >8-bit mantissa in multiple clock cycles . The second solution has a higher throughput that employs more hardware to perform the operations on the block of inputs within a single clock cycle.

## 6.2 Related Work

Jeong *et al.* (2022) supports a range of precisions from FP8 to FP32 with weights fetched from a Weight Storage (WS-SRAM) (Figure 37). The computations are performed near-memory and relayed back to Mantissa-Product storage (MP-SRAM) and Exponent-Sum storage (ES-CAM). These computations are performed over multiple cycles. The latency is dependent on the precision, so for higher precisions the latency is significantly higher.

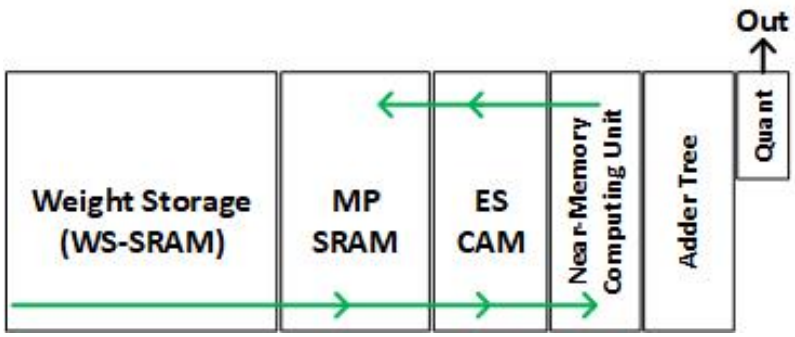



Figure 37. Block Diagram of the Jeong *et al.* (2022) Work

Tu *et al.* (2022) supports BFloat16 and FP32. They also share an architecture with fixed point adder trees which are accumulative, resulting in some area overhead gains (Figure 38). But it lacks tight integration of computation and memory and has low throughput.

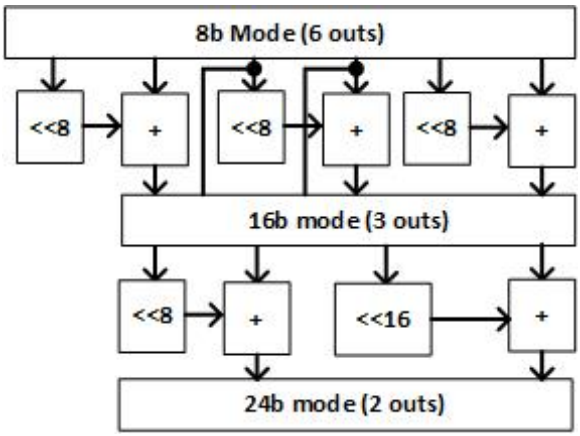



Figure 38. Block Diagram of the Tu *et al.* (2022) Work

Guo *et al.* (2023) uses a exponent comparator, to force a common exponent adjusting mantissa accordingly (Figure 39). This eliminates the need for a exponent memory and also allows for use of a fully fixed point adder tree allowing for accumulative adder trees.

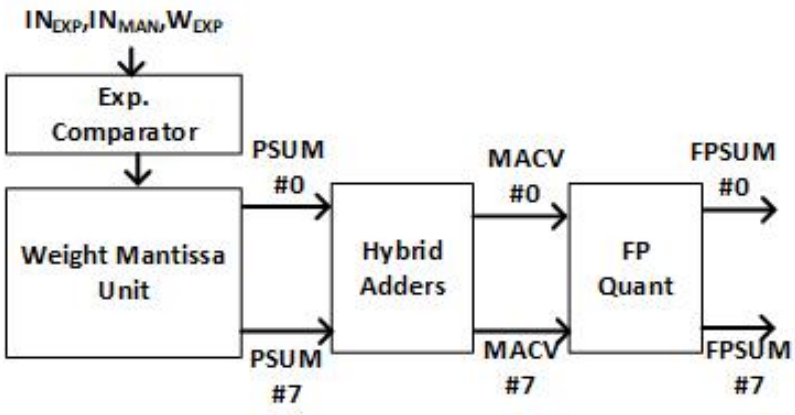



Figure 39. Block Diagram Description of the Guo *et al.* (2023) Work

## 6.3 Proposed Work

In this work, we draw inspiration from the three works referred to in the previous section to improve upon the FP-IMC work. We replace the FP adder trees with fixed-point adder trees to sum up the mantissa partial products, eliminating the need for separate adders for the exponents and the corresponding memory cells by incorporating these operations with appropriate adjustment of the mantissa in the adder trees. Further, we make three groups, <8-bit, 9-16 bit and >16-bit of mantissa lengths, for the seven floating point formats and make provision for computation of the mantissa products through three different groupings of the memory cells and appropriate use of three types of adder trees. The resulting architecture provides higher throughput, economy of hardware, improved energy efficiency and wider FP precision options

### 6.3.1 Supported Precision Modes

This work supports 7 modes, shown in (S,E,M) format:

1. FP8 (1,5,2): Considered a MiniFloat, a group of floating-point values constituting a small number of bits. Used in specific applications, where compactness and reduced complexity is preferred over accuracy.
2. FP16,(1,5,10): Also known as half-precision. It trades-off complexity in favour of more accuracy.
3. BF16,(1,8,7): BrainFloat16, developed by the team at Google Brain. This format offers a wider dynamic range in addition to the benefits offered by FP16. FP-32 values can be translated to BrainFloat16.

4. TF-32,(1,8,10): Tensor Float-32 introduced by NVIDIA to add more significand bits to BrainFloat16 format.
5. AP24,(1,7,16): Used in Radeon R300 and R420 GPUs.
6. PXR24,(1,8,15): Used by Pixar Animation Studios to convert FP32 to 24-bits as a compression tool.
7. FP32,(1,8,23): Widely popular IEEE-754 Floating-point format, also referred to as single precision. Offers a very high precision but suffers from implementation complexities.

The exponent length in these formats are:

- 5-bit in FP8 and FP16.
- 7-bit in AP24.
- 8-bit in FP16,TF-32, PXR24 and FP32.

The FP formats are categorized into three groups based on the mantissa length:

- Mantissa Group1 (MG1): This group includes modes with <8-bit mantissa, FP8 (2-bit) and FP16 (7-bit).
- Mantissa Group2 (MG2): This group includes modes with 9-16 bit mantissa, FP16 (10-bit), TF-32 (16-bit), AP24 (16-bit) and PXR24 (15-bit).
- Mantissa Group3 (MG3): This group includes modes with >16 bit mantissa, FP32 (23-bit).

While the operations on a block of inputs is performed in a single cycle for FP formats in MG1, we provide two solutions for the remaining higher precision modes. The first approach, **single macro design**, performs the operations on a single block of input data in multiple cycles, where the number of cycles is 1 in MG1, 2 in MG2

and 3 in MG3. In the second approach, **multi macro design** the operations on a block of input data are performed in a single cycle.

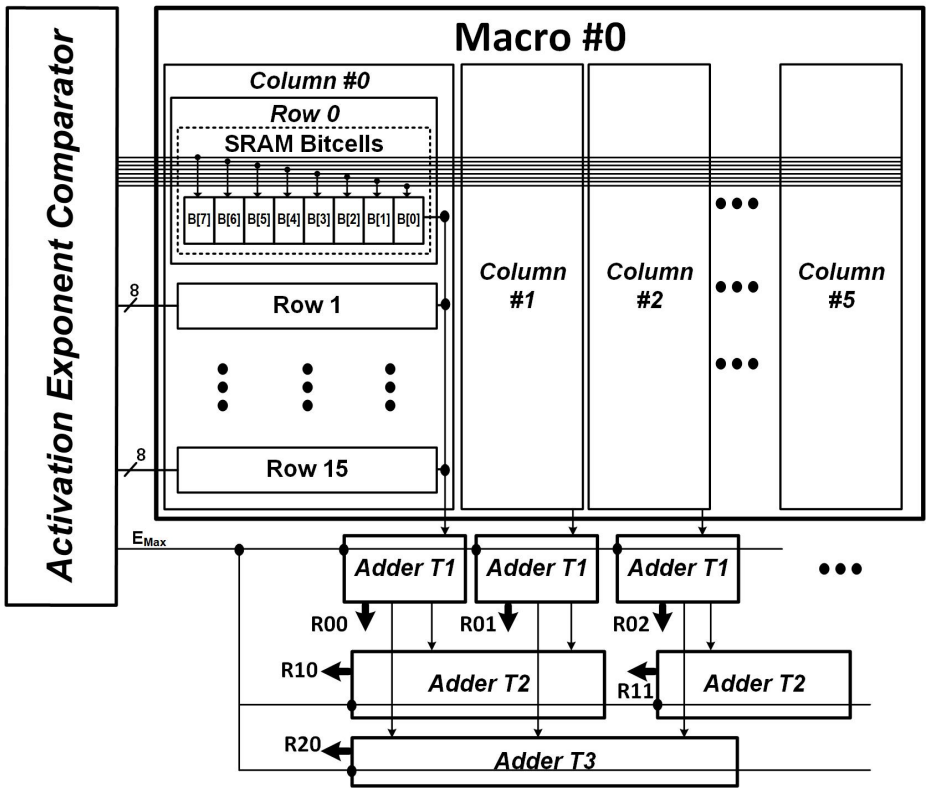



Figure 40. Block Diagram of the Macro Architecture

## 6.4 Organization of the Macros

The block diagram in Figure 40 shares the 16x8x6 **single macro design** architecture. The design follows a weight stationary approach. One macro comprises of 6 Memory Columns each holding 16 rows of 8-bit words. Each 8-bit word is stored and computed (bitwise-multiplication) over 8 bitcells. The mantissa products from the 16 rows are then accumulated in an Adder Tree. Three types of Adder trees are employed to support the FP formats in MG1, MG2 and MG3. If we consider MG1 modes, the

IMC Macro can perform the MAC computation for a fully connected layer of ANN with 16 input nodes and 6 output nodes. The Multi-Macro design can perform MAC computations for 16 activation input and 36 outputs in MG1 FP format.

In the following section we discuss the design in detail describing the operations for the higher precision FP operations.

## 6.5 The Single Macro Design

The activation exponent comparator handles the exponent bits of the activation and outputs the adjusted activation mantissa. Since the memory only deals with the mantissa, only one type of memory bitcell is used as opposed to the 3 types used in the earlier FP-IMC work.

This design also incorporates the cumulative adder tree shared in Tu *et al.* (2022) in the form of the three adder tree types:

- AdderTree type1 (AT1) handles MG1 mantissa precision,
- AdderTree type2 (AT2) handles MG2 mantissa precision and
- AdderTree type3 (AT3) handles MG3 mantissa precision

The normalized outputs for MG1 are obtained from the AT1 as R0X. Here, X represents the tree number. Similarly, the normalized outputs for MG2 are obtained from AT2 as R1X and from AT3 as R2X respectively.

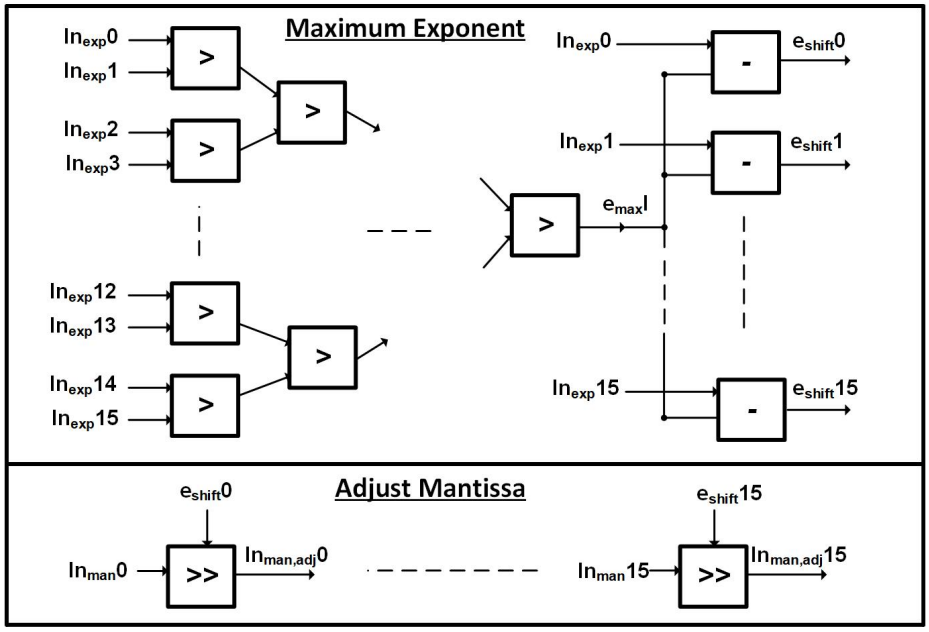



Figure 41. Description of the Exponent Comparator

### 6.5.1 Activation Exponent Comparator

An Activation Exponent Comparator is used in the design to determine the maximum exponent among the 16 Activation words sent over to each column. Figure 41 shows the Activation Exponent Comparator. The comparator determines the maximum exponent, $e_{max}$. Then the difference between the $e_{max}$ and exponent for each of the words is obtained. This allows for adjusting the mantissa accordingly.

### 6.5.2 Bitcell

The bitcell is akin to the T3 bitcell from FP-IMC. As the mantissa range varies quite a bit, there are updates to support this range. While the T3 bitcell in FP-IMC can perform a 2-bit multiplication, using additional 8Ts as a pair of NANDs. The

38-T bitcell, in addition to the 6T SRAM, uses 8 sets of NANDs for the eight bit-bit multiplications. This bit-bit multiplication output from each bitcell gives a 8-bit partial product. A weighed sum of these partial products and accumulation over the 16 words, one in each Memory Column row, is performed inside the Adder trees.

### 6.5.3 AdderTrees

As the mantissa product is in fixed point, the accumulation can be cumulative. The accumulation of the 8-bit mantissa product is fairly simple and is done inside the AdderTree type1 shown in Figure 42.

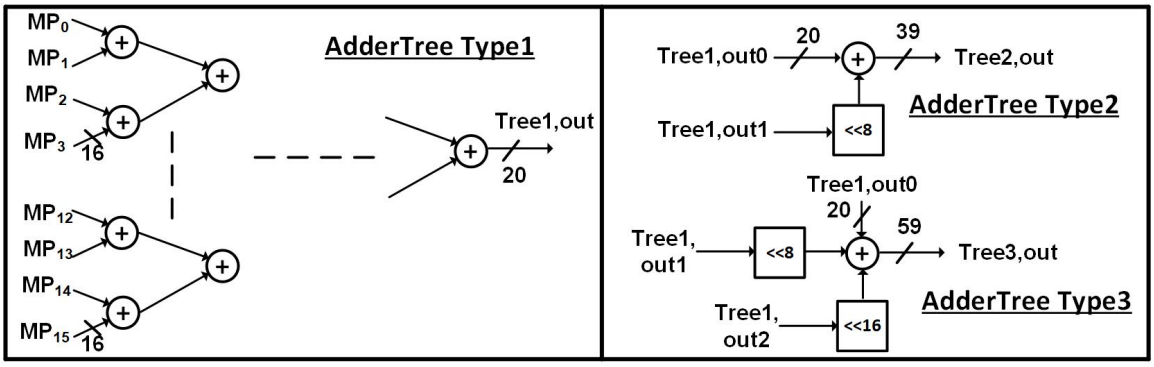



Figure 42. (Left) Description of AdderTree Type1 (AT1) (Right) Use of AT1 outputs to get AT2 and AT3 Outputs

One set, equal to the number of columns, of 8-bit mantissa product accumulations is performed every clock cycle to obtain Tree1,out. For 16-bit and 24-bit mantissa product accumulation, the activation and weights are broken down into 8-bit chunks. The chunks of weights are stored over multiple columns (2 for MG2 and 3 for MG3). The chunks of activations are relayed over multiple clock cycles (1,2 and 3 for MG1, MG2 and MG3 respectively). Each chunk product accumulation produces a Tree1,out. These Tree1,outs can then be accumulated over 2 clock cycles inside the AT2 to

obtain 16-bit mantissa product accumulation. Similarly, the 24-bit mantissa product is accumulated over 3 clock cycles inside the AT3.

### 6.5.4 Normalization Modules

There are three types of Normalizers (NT1,NT2 and NT3) used inside the adder trees AT1, AT2 and AT3 respectively. A generalized Normalizer is shared in Figure 43, detailing the common functionalities each of the normalizers perform but for different precisions. The MAC output from the adder trees are fed into the normalizer. The leading '1' is detected from the mantissa of this input, and accordingly the exponent is adjusted. Based on this leading '1', M-bits of mantissa are taken, exponents computed and the sign collected to form the FP MAC outputs.

The normalizers, Normalizer type1(NT1), Normalizer type2 (NT2), and Normalizer type3 (NT3) are used for MG1, MG2 and MG3 formats respectively.

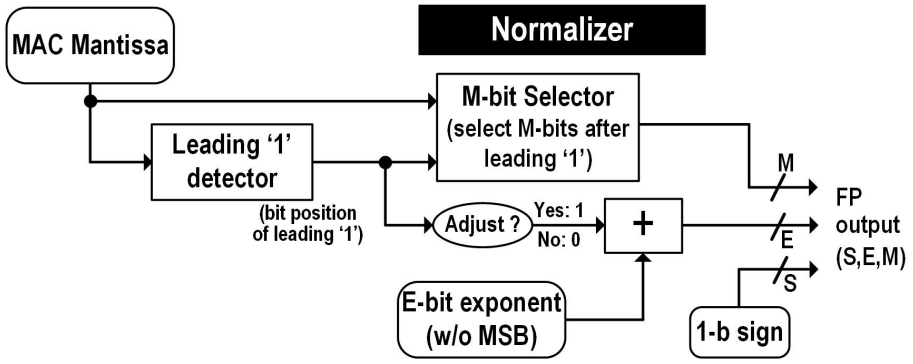



Figure 43. A Generalized Normalization Module for each FP Mode following the (S,E,M) Notation.

## 6.6 The Multiple Macro Design Architecture

The **single macro design** architecture requires 2 and 3 clock cycles to implement the MAC operations in MG2 and MG3 FP formats through repeated use of the functional modules. In this section, we present the **multiple macro design** organized for single cycle the FP operation for all three mantissa group modes. With the multiple macros organized together, the bit-wise mantissa products are available at once, which can then be accumulated to obtain the MAC. All of this is done in a single cycle.

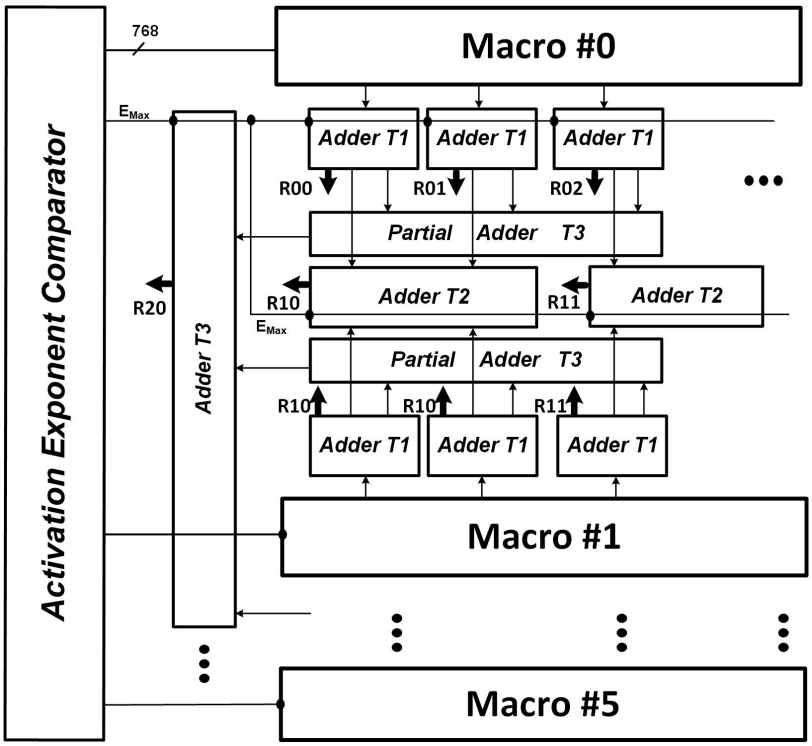



Figure 44. Architecture of the Single Cycle Approach

Six Columns each holding 16 words of 8-bit mantissa form a macro as previously discussed in section 6.5. In the **multiple macro design**, we place 6 such macros,

organizing them vertically, making the memory size six times larger than that of the **single macro design**.

To perform the multiplication in the MG2 and MG3 FP formats within a single cycle we need to suitably map 8-bit chunks of the weight mantissa and activation input mantissa in the memory as elaborated in section 6.8.

## 6.7 Avoiding Idling

The motive behind implementing 6 columns of the memory in the horizontal and vertical directions is inducted to avoid idling of some memory columns depending on the mode. To illustrate this we take the example of different macro sizes ranging from 3x1 to 6x1 memory column array in Figure 45. In the figure chunks of the same word are depicted in the same color.

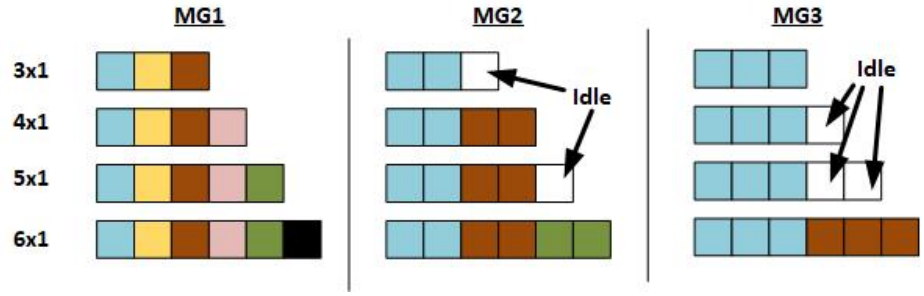



Figure 45. Idling of the Memory Column Based on the Word Allocation in Memory

For MG1 modes, one chunk can be allocated to single memory column, so all the memory columns in the example macros can hold a unique word. In the MG2 modes, two chunks are to be allocated in adjacent memory columns, leaving a unused memory column in the 3x1 and 5x1 example macros. Similarly for MG3 modes, three chunks are to be allocated in adjacent memory columns, leaving unused memory columns in

the 4x1 and 5x1 macro sizes. To effectively use the columns without any idling, the array size should be a common multiple of 1,2 and 3. The lowest value meeting this criteria is 6. The same reasoning is applied in the choice of 6 macros.

## 6.8 Example Multiplication Operation

In this section we share an example of 24-bit mantissa multiplication operation. This idea can be extended to the other modes by appropriately allocating the word chunks in memory.

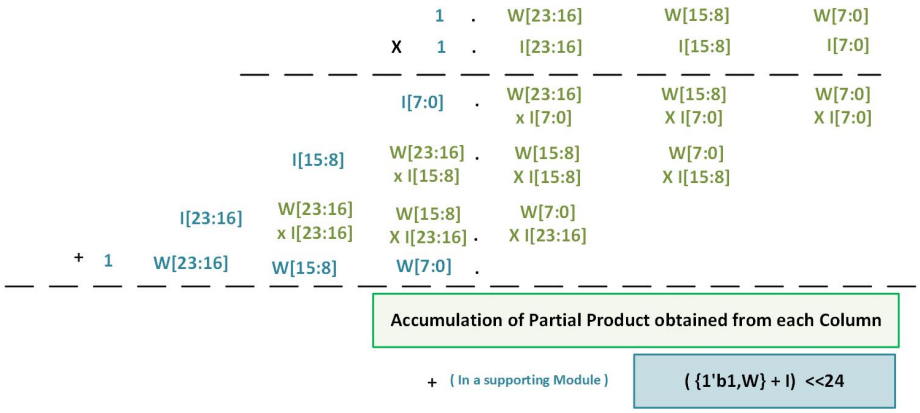



Figure 46. Multiplication Operation on 24-bit Mantissa

In the example in Figure 46, the 24-bit mantissas of the weight and input(activation) are broken down into chunks of 8-bit. While the products with the multiplicands as input and weights are performed in the bitcells, the products with the leading '1's and either an input mantissa chunk or a weight mantissa chunk are performed in supporting modules.

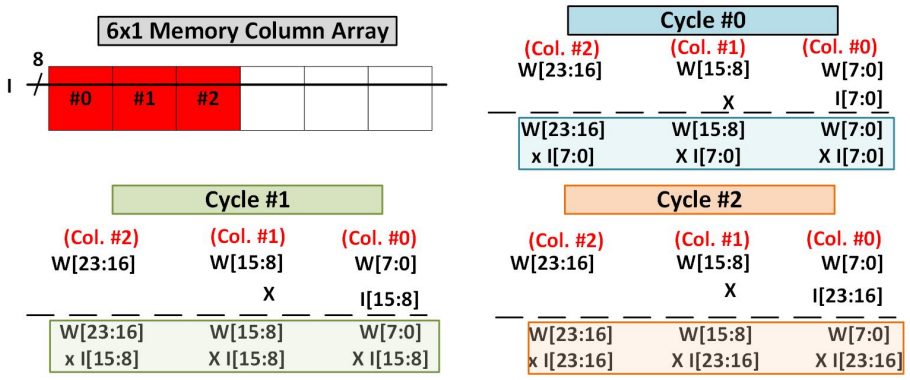



Figure 47. Multiplication Operation on 24-bit Mantissa for the Single Macro Design

For the **single macro design**, shown in Figure 47, the multiplication is organised as follows. The 24-bit weight mantissa broken down into 3 chunks of 8-bits are stored in 3 adjacent memory columns. The input mantissa broken into 3 chunks of 8-bits are then fed into the macro one at a time over subsequent cycles. So all of the partial products are available over three cycles.

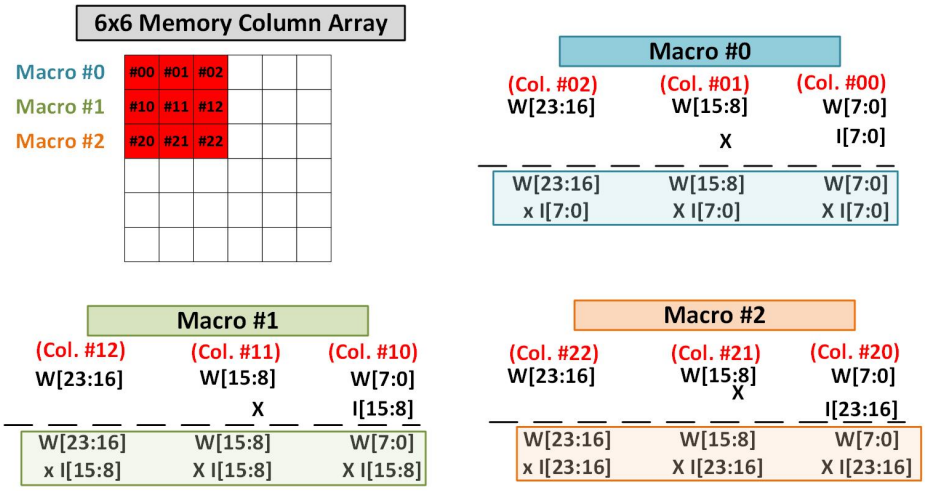



Figure 48. Multiplication Operation on 23b mantissa in the FP32 Mode for Single Cycle Approach

For the **multiple macro design**, shown in Figure 48, the input words mantissa chunks is fed into one macro in the MG1 modes, into two adjacent macros in the

MG2 modes, and three adjacent macros in the MG3 modes. The weights stored in the macro are duplicated once to another macro for the MG2 modes, and duplicated one more time for MG3 modes. All of the partial products are available in a single cycle.

## 6.9 Simulation Experiments and Results

The two approaches were implemented in 28nm tech CMOS. While the physical design portion of the **single macro design** is completed, with post-APR static-timing-analysis (STA) conducted. The experiments with the **multiple macro design** are still in the preliminary stages. To make a fair comparison, only the post-synthesis Primetime power numbers are reported for both approaches in Figure 49.

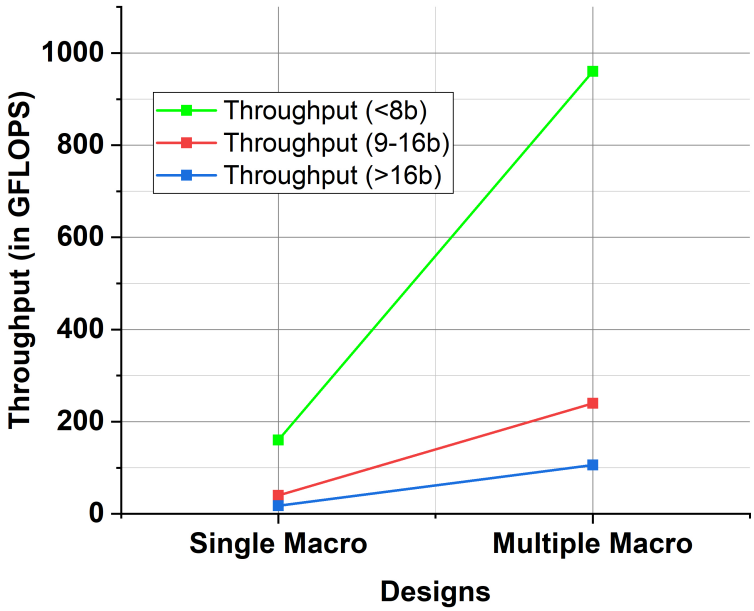



Figure 49. Comparing the Two Approaches: Single Macro Design and Multiple Macro Design.

The **multiple macro design** has a larger memory implementation to support simultaneous executions of the partial products, adding significant hardware. The area for the **multiple macro design** comes out at >6× the **single macro design** area but has a 6× higher throughput. The power scales by a similar margin too, meaning both the approaches offer very similar energy-efficiency.

TABLE III
COMPARISON WITH THE FP-IMC WORK

| | FP-IMC work | This work (Single Macro Design) |
|---|---|---|
| **Technology** | 28nm | 28nm |
| **Memory** | 4Kb | 0.75 Kb |
| **CIM Bitcell** | 8T/14T | 38T |
| **FP Precision** | FP8/BF8 | FP8-FP32 (7modes) |
| **Operational Frequency** | 484.7 MHz (@0.9V) | 833 MHz (@ 0.9 V) |
| **Area ($mm^2$)** | 1.2x1.3 | 0.42x0.88 (macro) |
| **Normalized Peak Throughput (GFLOPS/Kb)** | 121.5 (FP8) | 93.2 (FP8@484.7MHz) |
| **Energy Efficiency (TFLOPS/W)** | 10.02 (FP8) | 4.09 (FP8@484.7 MHz) |
| **Normalized Energy Efficiency (TFLOPS/W/Kb)** | 2.50 (FP8) | 5.46 (FP8@484.7 MHz) |

## 6.10 Comparison to FP-IMC Work

We compare the results of the post-APR Primetime report for the **single macro design** of this work with the FP-IMC work on-chip results and share it in Table III. This work implements the memory at a smaller scale. So, to make a fair comparison

the normalized throughput and energy-efficiency are shared in the table. The FP-IMC work on-chip implementation tops out at 484.7 MHz at 0.9V, the **single macro design** in this work, can operate at up to 833MHz. To make a fair comparison the throughput and energy-efficiency numbers are reported for this work at 483.7 MHz for FP8 mode. This work shows 2.1× normalized energy-efficiency compared to the FP-IMC work.

## 6.11 Summary

In this work, the FP-IMC work was extended to support many higher precision Floating-point modes. Additionally, architectural changes were proposed to improve upon the energy-efficiency of the FP-IMC design. Finally, two approaches of the improved work were implemented and the benefits and drawbacks of each were highlighted. This re-configurable IMC work supporting 7 FP precision modes including FP8, offers over 2.1 × normalized energy-efficiency compared to the FP-IMC work.

# Chapter 7

# CONCLUSION

To combat the extensive number of memory accesses required in conventional computing hardware for the compute-intensive ML tasks, we use in-memory computing for AI/ML hardware accelerators. In this dissertation, we focus on two sets of ML tasks: the $k$NN computation in the kNN accelerator work and floating point MAC operations for high accuracy DNNs plane computational accelerators in the rest of the works.

In this dissertation, we presented a simulation framework including different core design parameters and characterized the effect of variation source on network-level accuracy. Picking an optimal ADC precision however comes down to the noise sensitivity of the model. The model was also verified over 3 different DNN Networks.

Next, we present a $k$NN accelerator leveraging the fact that the XNOR-SRAM IMC XAC output serves a distance information between the input and the weight. A digital sorter is designed to process this distance information and output the $k$NN outputs. The Accelerator can process up to 17.9 M query Vectors/s while consuming 18.4 mW. We observe a 4.8x better energy/query vector compared to prior works. However the design is not scaling friendly in terms of power consumption.

Further, we presented a novel floating-point precision IMC macro where the float-pointing computation is immersed into the weight memory storage. The proposed FP-IMC supports two FP8 configurations of (1) normal 8-bit floating-point (FP8) precision with 1-bit sign, 5-bit exponent, and 2-bit mantissa (1-5-2), and (2) 8-bit block floating point (BF8) precision with a shared exponent among 64-element weight

tensor . Our proposed FP-IMC macro was implemented in a prototype chip in 28nm CMOS, and achieves 12.1 TFLOPS/W for FP8 and 66.6 TFLOPS/W for BF8.

Finally, we move towards improving the FP IMC work by supporting multiple higher precision modes with 8 bit to 32 bit FP formats. With additional changes made to the architecture to improve energy-efficiency for minimal accuracy degradation. For the formats with >8-bit mantissa two different solutions are provided: (i) to support the operations on a data block in multiple cycle using minimal hardware and (ii) to support the operations on a data block in single cycle. The second solution requires additional hardware but provides higher throughput. This work shows at least a 2.1 $\times$ normalized energy efficiency compared to the FP-IMC work.

In respect of further works, the immediate further work that can be carried out is to implement the RFP-IMC in a chip and test its performance. It will also be worthwhile to investigate the following:

1. Broaden the IMC parameter study to encompass eNVMs, which come with their own set of variation noises.
2. Using the noise injection used in the python framework in the training phase to develop noise-robust DNNs and comparing the results against regularly trained DNNs.
3. Extending the RFP-IMC work to multiplications in sets of smaller chunks from 8 to 4 or 2, to achieve better utilization of the hardware.
4. Supporting double and quadruple precision for applications such as scientific computing.
5. The IMC bit cell used in the work, XNOR-SRAM has possibility for tweaking and use to get other innovative products. It also has the potential for use in In-Cache AI/ML accelerators.

In summary, this dissertation comprehensively discusses novel architectures to advance the implementation of DNN networks for incorporating AI/ML accelerators in edge-devices

# REFERENCES


Bremler-Barr, A., Y. Harchol, D. Hay and Y. Hel-Or, "Ultra-fast similarity search using ternary content addressable memory", in "Proceedings of the 11th International Workshop on Data Management on New Hardware", pp. 1–10 (2015).

Chih, Y.-D., P.-H. Lee, H. Fujiwara, Y.-C. Shih, C.-F. Lee, R. Naous, Y.-L. Chen, C.-P. Lo, C.-H. Lu, H. Mori *et al.*, "16.4 an 89tops/w and 16.3 tops/mm 2 all-digital sram-based full-precision compute-in memory macro in 22nm for machine-learning edge applications", in "2021 IEEE International Solid-State Circuits Conference (ISSCC)", vol. 64, pp. 252–254 (IEEE, 2021).

Choi, W., K. Jeong, K. Choi, K. Lee and J. Park, "Content addressable memory based binarized neural network accelerator using time-domain signal processing", in "Proceedings of the 55th Annual Design Automation Conference", pp. 1–6 (2018).

Dong, Q., M. E. Sinangil, B. Erbagci, D. Sun, W.-S. Khwa, H.-J. Liao, Y. Wang and J. Chang, "15.3 a 351tops/w and 372.4 gops compute-in-memory sram macro in 7nm finfet cmos for machine-learning applications", in "2020 IEEE International Solid-State Circuits Conference-(ISSCC)", pp. 242–244 (IEEE, 2020).

Guo, A., X. Si, X. Chen, F. Dong, X. Pu, D. Li, Y. Zhou, L. Ren, Y. Xue, X. Dong *et al.*, "A 28nm 64-kb 31.6-tflops/w digital-domain floating-point-computing-unit and double-bit 6t-sram computing-in-memory macro for floating-point cnns", in "2023 IEEE International Solid-State Circuits Conference (ISSCC)", pp. 128–130 (IEEE, 2023).

Hong, I., J. Park, G. Kim, J. Oh and H.-J. Yoo, "A 125,582 vector/s throughput and 95.1% accuracy ann searching processor with neuro-fuzzy vision cache for real-time object recognition", in "2013 Symposium on VLSI Circuits", pp. C184–C185 (IEEE, 2013).

Imani, M., Y. Kim and T. Rosing, "Nngine: Ultra-efficient nearest neighbor accelerator based on in-memory computing", in "2017 IEEE International Conference on Rebooting Computing (ICRC)", pp. 1–8 (IEEE, 2017).

Jeong, S., J. Park and D. Jeon, "A 28nm 1.644 tflops/w floating-point computation sram macro with variable precision for deep neural network inference and training", in "ESSCIRC 2022-IEEE 48th European Solid State Circuits Conference (ESSCIRC)", pp. 145–148 (IEEE, 2022).

Jia, H., H. Valavi, Y. Tang, J. Zhang and N. Verma, "A programmable heterogeneous microprocessor based on bit-scalable in-memory computing", IEEE Journal of Solid-State Circuits **55**, 9, 2609–2621 (2020).

Jiang, Y., J. Kang and X. Wang, "Rram-based parallel computing architecture using k-nearest neighbor classification for pattern recognition", Scientific reports **7**, 1, 45233 (2017).

Jiang, Z., S. Yin, J.-S. Seo and M. Seok, "C3sram: An in-memory-computing sram macro based on robust capacitive coupling computing mechanism", IEEE Journal of Solid-State Circuits **55**, 7, 1888–1897 (2020).

Jouppi, N. P., D. H. Yoon, M. Ashcraft, M. Gottscho, T. B. Jablin, G. Kurian, J. Laudon, S. Li, P. Ma, X. Ma *et al.*, "Ten lessons from three generations shaped google's tpuv4i: Industrial product", in "2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)", pp. 1–14 (IEEE, 2021).

Kang, M., Y. Kim, A. D. Patil and N. R. Shanbhag, "Deep in-memory architectures for machine learning–accuracy versus efficiency trade-offs", IEEE Transactions on Circuits and Systems I: Regular Papers **67**, 5, 1627–1639 (2020).

Kaplan, R., L. Yavits and R. Ginosar, "Prins: Processing-in-storage acceleration of machine learning", IEEE Transactions on Nanotechnology **17**, 5, 889–896 (2018).

Kaul, H., M. A. Anders, S. K. Mathew, G. Chen, S. K. Satpathy, S. K. Hsu, A. Agarwal and R. K. Krishnamurthy, "14.4 a 21.5 m-query-vectors/s 3.37 nj/vector reconfigurable k-nearest-neighbor accelerator with adaptive precision in 14nm tri-gate cmos", in "2016 IEEE International Solid-State Circuits Conference (ISSCC)", pp. 260–261 (IEEE, 2016).

Lee, V. T., J. Kotalik, C. C. Del Mundo, A. Alaghi, L. Ceze and M. Oskin, "Similarity search on automata processors", in "2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)", pp. 523–534 (IEEE, 2017).

Seo, J.-s., J. Saikia, J. Meng, W. He, H.-s. Suh, Y. Liao, A. Hasssan, I. Yeo *et al.*, "Digital versus analog artificial intelligence accelerators: Advances, trends, and emerging designs", IEEE Solid-State Circuits Magazine **14**, 3, 65–79 (2022).

Tu, F., Y. Wang, Z. Wu, L. Liang, Y. Ding, B. Kim, L. Liu, S. Wei, Y. Xie and S. Yin, "A 28nm 29.2 tflops/w bf16 and 36.5 tops/w int8 reconfigurable digital cim processor with unified fp/int pipeline and bitwise in-memory booth multiplication for cloud deep learning acceleration", in "2022 IEEE International Solid-State Circuits Conference (ISSCC)", vol. 65, pp. 1–3 (IEEE, 2022).

Valavi, H., P. J. Ramadge, E. Nestler and N. Verma, "A 64-tile 2.4-mb in-memory-computing cnn accelerator employing charge-domain compute", IEEE Journal of Solid-State Circuits **54**, 6, 1789–1799 (2019).

Wang, N., J. Choi, D. Brand, C.-Y. Chen and K. Gopalakrishnan, "Training deep neural networks with 8-bit floating point numbers", Advances in neural information processing systems **31** (2018).

Wu, P.-C. *et al.*, "A 22nm 832Kb hybrid-domain floating-point SRAM in-memory-compute macro with 16.2-70.2 TFLOPS/W for high-accuracy AI-edge devices", in "IEEE ISSCC", (2023).

Yin, S., Z. Jiang, J.-S. Seo and M. Seok, "Xnor-sram: In-memory computing sram macro for binary/ternary deep neural networks", IEEE Journal of Solid-State Circuits **55**, 6, 1733–1743 (2020).

Zhang, B., J. Saikia, J. Meng, D. Wang, S. Kwon, S. Myung, H. Kim, S. J. Kim, J.-s. Seo and M. Seok, "A 177 tops/w, capacitor-based in-memory computing sram macro with stepwise-charging/discharging dacs and sparsity-optimized bitcells for 4-bit deep convolutional neural networks", in "2022 IEEE Custom Integrated Circuits Conference (CICC)", pp. 1–2 (IEEE, 2022).

# APPENDIX

## A. LIST OF PUBLICATIONS

This dissertation includes works presented in published co-authored papers. Permissions have been obtained from all co-authors to share the works from these papers. Here is the list of all the co-authored papers:

1. **Jyotishman Saikia**, Shihui Yin, Zhewei Jiang, Mingoo Seok, and Jae-sun Seo, "K-nearest neighbor hardware accelerator using in-memory computing SRAM," IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED), 2019.
2. **Jyotishman Saikia**, Shihui Yin, Sai Kiran Cherupally, Bo Zhang, Jian Meng, Mingoo Seok, and Jae-sun Seo, "Modeling and Optimization of SRAM-based In-Memory Computing Hardware Design," IEEE Design, Automation & Test in Europe (DATE), February 2021.
3. **Jyotishman Saikia**, Injune Yeo, Amitesh Sridharan, Shreyas Venkatramanaiah, Deliang Fan, Jae-sun Seo, "FPIMC: A 28nm All-Digital Configurable Floating-Point In-Memory Computing Macro," IEEE European Solid State Devices and Circuits Conference (ESSCIRC), September 2023.
4. Bo Zhang, **Jyotishman Saikia**, Jian Meng, Dewei Wang, Soonwan Kwon, Sungmeen Myung, Hyunsoo Kim, Sang Joon Kim, Jae-sun Seo, and Mingoo Seok, "A 177 TOPS/W, Capacitor-based In-Memory Computing SRAM Macro with Stepwise-Charging/Discharging DACs and Sparsity-Optimized Bitcells for 4-Bit Deep Convolutional Neural Networks," IEEE Custom Integrated Circuits Conference (CICC), April 2022.
5. Jae-sun Seo, **Jyotishman Saikia**, Jian Meng, Wangxin He, Han-sok Suh, Yuan Liao, Ahmed Hasssan, and Injune Yeo, "Digital Versus Analog Artificial Intelligence Accelerators: Advances, trends, and emerging designs." IEEE Solid-State Circuits Magazine, vol. 14, no. 3, pp. 65-79, Summer 2022.
6. **Jyotishman Saikia**, Shihui Yin, Zhewei Jiang, Mingoo Seok, and Jae-sun Seo, "K-nearest neighbor hardware accelerator using in-memory computing SRAM," poster presentation, SRC TECHCON, 2019.
7. **Jyotishman Saikia**, Shihui Yin, Sai Kiran Cherupally, Bo Zhang, Jian Meng, Mingoo Seok, and Jae-sun Seo, "Modeling and Optimization of SRAM-based In-Memory Computing Hardware Design," paper presentation, SRC TECHCON, 2021.