DEVELOPMENT OF PROCESS, VOLTAGE AND TEMPERATURE VARIATION AWARE HIGHLY ENERGY-EFFICIENT DEEP NEURAL NETWORKS WITH HIGH INFERENCE ACCURACY FOR INTERNET-OF-THINGS APPLICATIONS

by
UMUT BARUT

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfilment of
the requirements for the degree of Master of Science

Sabancı University
July 2024

DEVELOPMENT OF PROCESS, VOLTAGE AND TEMPERATURE VARIATION AWARE HIGHLY ENERGY-EFFICIENT DEEP NEURAL NETWORKS WITH HIGH INFERENCE ACCURACY FOR INTERNET-OF-THINGS APPLICATIONS

Approved by:

Asst. Prof. Ömer Ceylan .....
(Thesis Supervisor)

Prof. Yaşar Gürbüz .....

Asst. Prof. Atilla Uygur .....

Date of Approval: July 24, 2024

ABSTRACT

DEVELOPMENT OF PROCESS, VOLTAGE AND TEMPERATURE VARIATION AWARE HIGHLY ENERGY-EFFICIENT DEEP NEURAL NETWORKS WITH HIGH INFERENCE ACCURACY FOR INTERNET-OF-THINGS APPLICATIONS

UMUT BARUT

EE, M.Sc. THESIS, JULY 2024

Thesis Supervisor: Asst. Prof. Ömer Ceylan

Keywords: internet-of-things, deep neural network accelerator, energy efficiency, edge computing, probabilistic timing error model, PVT variation aware voltage underscaling

Artificial intelligence (AI models have improved with advancements in hardware, processing large datasets crucial for daily applications. The Internet-of-Things (IoT) enhances AI usage but brings challenges like bandwidth overhead, latency, and cyber threats. Edge computing and fast, energy-efficient hardware, like application-specific integrated circuit DNN accelerators, are essential solutions for IOT devices.

To further enhance energy efficiency, voltage reduction on power supplies is a viable method, despite causing timing errors that DNNs might tolerate due to their inherent nature. Extensive MAC operations also lead to significant switching activity, potentially generating noise on the chip's power lines. In this thesis, process voltage temperature (PVT) aware probabilistic timing error model is developed and demonstrated to find error probability due to voltage and temperature noise. By utilizing this model, bit error probability can be calculated without relying on the time-consuming Monte Carlo (MC) simulations, thus the analysis time of the digital hardware is significantly reduced. By observing the inference accuracy of a DNN model through the introducing of bit errors to its layers, designers can optimize power consumption by employing dynamic voltage scaling based on layer importance.

A 16x16 systolic MAC array accelerator was designed using 65nm CMOS technology to verify the model. Single MAC unit is analyzed for timing error probability and result is compared with MC simulations. The model demonstrated decent accuracy and was approximately 808 times faster than MC simulations (1500 sample), allowing for rapid observation of voltage reduction effects on the accelerator in terms of timing error probability.

ÖZET

NESNELERİN İNTERNETİ UYGULAMALARI İÇİN GERİLİM, SICAKLIK VE ÜRETİMSEL SAPMALARI DİKKATE ALARAK YÜKSEK ENERJİ VERİMLİLİĞİ VE ÇIKARIM DOĞRULUĞUNA SAHİP DERİN ÖĞRENME AĞLARI GELİŞTİRİLMESİ

UMUT BARUT

EE, YÜKSEK LİSANS TEZİ, TEMMUZ 2024

Tez Danışmanı: Dr. Öğr. Üyesi Ömer Ceylan

Anahtar Kelimeler: Nesnelerin interneti, derin öğrenme ağı, enerji verimliliği, uçta yapay zeka, olasılıksal zamanlama hata modeli, zamanlama hata olasılığı, derin öğrenme ağı geliştirme platformu, besleme geriliminin düşürülmesi, PVT sapmalarını dikkate alarak besleme gerilimini düşürme

Yapay zeka modelleri, donanımda kaydedilen ilerlemelerle birlikte gelişerek günlük uygulamalar için büyük veri kümelerini işlemek konusunda önemli hale geldi. Nesnelerin interneti, yapay zeka kullanımını artırsa da bant genişliği, gecikme ve siber tehditler gibi zorluklar ortaya çıkarır. Yerinde hesaplama ve hızlı, enerji verimli donanımlar, özellikle tümleşik devre yapay zeka hızlandırıcıları, nesenelerin interneti cihazları için önemli çözümler arasında yer alır.

Enerji verimliliğini daha da artırmak için voltaj düşürme yöntemi uygulanabilir, bu durum zamanlama hatalarına yol açabilir fakat derin sinir ağlarının doğası gereği bu hatalar tolere edebilebilir. Yoğun çarpma ve toplama işlemleri önemli ölçüde girdi ve çıktılarda değişikliğe sebep olduğundan, çipin güç hatlarında gürültü oluşturabilir. Bu tezde, voltaj ve sıcaklık gürültülerinden kaynaklanan hata olasılığını belirlemek için, işlem voltajı, sıcaklık ve üretimsel sapmalardan kaynaklı durumları hesaba katan olasılıksal hata modeli geliştirilmiş ve çalıştığı gösterilmiştir. Bu model kullanılarak, zaman alıcı Monte Carlo simülasyonlarına başvurmadan bit hata olasılığı hesaplanabilir, böylece dijital donanımın analizi için harcanan süre önemli ölçüde azaltılmış olur. Tasarımcılar, DNN modelinin katmanlarına bit hataları ekleyerek

çıkarım doğruluğu gözlemleyip, katman önemine göre dinamik voltaj ölçeklendirmesi uygulayarak güç tüketimini optimize edebilirler.

Modeli doğrulamak için 65nm CMOS teknolojisi kullanılarak 16x16 boyutunda bir sistolik çarpma ve toplama dizisi, hızlandırıcısı tasarlanmıştır. Tek bir çarpma ve toplama birimi zamanlama hata olasılığı açısından analiz edilmiş ve sonuçlar Monte Carlo simülasyonları ile karşılaştırılmıştır. Model makul bir doğruluk göstermiş ve Monte Carlo simülasyonlarından (1500 örnek) yaklaşık 808 kat daha hızlı bir şekilde hata oranı hesaplaması yapmıştır. Bu da hızlandırıcı üzerindeki voltaj düşürme etkilerinin zamanlama hata olasılığı açısından hızlı bir şekilde gözlemlenmesine olanak tanımıştır.

ACKNOWLEDGEMENTS

First of all, I would like to thank to my supervisor Asst. Prof. Ömer Ceylan for his invaluable advice, support and guidance during my master's studies at Sabancı University. I would not be at this stage in my career without the contributions, support, and endless motivation that he showed.

In addition to my thesis advisor, I would like to thank my thesis jury members, Prof. Yaşar Gürbüz and Asst. Prof. Atilla Uygur for their valuable time to spare in my thesis committee and for their precious comments, advice, and feedback. And also I would like to thank Prof. Emre Salman for guidance and contributions to project that I worked.

I thank my friends and colleagues at the Sabancı University Microelectronics Research Group (SUMER), old and new members, especially Tahsin Alper Özkan and Serhan Özboz for their friendship which makes the working environment comfortable for me and keep me motivated. I also thank the old and current laboratory specialist, Ali Kasal and Enver Çabuk for their support and help and friendship.

I am grateful to many friends for their friendship and support throughout the time in university, especially Alperen Doğan and Can Alper Önol.

Most importantly, I would like to thank my parents Cennet and Mustafa, my brother Süleyman for their endless support and love.

And lastly, I would like to thank UZAKRIDESERIES for organising painful long distance cycling events called UZAK. That kept me motivated for cycling, eventually keeping myself healthy both physically and mentally, throughout the time in university.

This work was supported by Turkish Scientific and Technology Research Institution (TUBITAK) under grant 121E217, "Development of Process, Voltage and Temperature Variation Aware Highly Energy-Efficient Deep Neural Networks with High Inference Accuracy for Internet-of-Things Applications", that made this thesis possible.

to my beloved family...

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AI Artificial Intelligence iv

ASIC Application Specific Integrated Circuit 12

AVDD Analog Power Supply Voltage 68

CNN Convolutional Neural Network 5, 49

CPU Central Processing Unit 9

CTS Clock Tree Synthesis 58

CUDA Compute Unified Device Architecture 10

DFF D-Flip-Flops 26

DNN Deep Neural Network 14, 17, 18, 48

DRC Design Rule Check 58

EDA Electronic Design Automation 18, 57, 80

ESD Electrostatic Discharge 68

GPU Graphics Processing Unit 9

HDL Hardware Description Language 57, 59

I/O Input/Output 9, 67, 72, 73

IC Integrated Circuits 12

IoT Internet of Things iv

LLM Large Language Models ..... xiii, 6

LSB Least Significant Bit 61

LSTM Long Short-Term Memory 5

LVS Layout versus Schematic 58

MAC Multiply and Accumulate 8, 16, 30, 40, 51

MC Monte Carlo ..... iv, 16, 20, 27

MSB Most Significant Bit 43, 47, 61

PCB Printed Circuit Board 19, 58

PDF Probability Density Function 19, 24, 25

POC Power on Control 68

PVT Process Voltage Temperature ..... iv, 14, 15, 16, 17

RAM Random Access Memory 56

Relu Rectified Linear Unit 49, 61

RF Register File 64, 73, 74

RNN Recurrent Neural Network 5

SDF Standard Delay Format 76

SRAM Static Random Access Memory 64

STA Static Timing Analysis 20, 37, 42

TPU Tensor Processing Unit 5, 12

VDD Digital Power Supply Voltage 68

1. INTRODUCTION

1.1 Artificial Intelligence

Artificial Intelligence (AI) represents one of the most transformative technologies of the 21st century, poised to reshape industries, economies, and societies. At its core, AI involves the development of computer systems capable of performing tasks that typically require human intelligence. These tasks include learning, reasoning, problem-solving, perception, language understanding, and decision-making.

The origins of AI can be traced back to the mid-20th century when pioneers like Alan Turing and Arthur Samuel laid the foundational concepts. Turing's seminal question, "Can machines think?" and his development of the Turing Test sparked significant interest and research into creating intelligent systems (Turing (1950)). Samuel's early experiments with game-playing programs demonstrated the potential for computers to improve performance over time through experience (Samuel (1959)). Since then, AI has progressed through several waves of innovation and setbacks, each marked by breakthroughs and the eventual realization of previously theoretical possibilities (Russell, Norvig & Davis (2010)).

AI can be broadly categorized into narrow AI and general AI. Narrow AI, also known as weak AI, is designed to perform specific tasks such as speech recognition, image classification, or recommendation systems. These systems have become increasingly sophisticated, driven by techniques such as machine learning, deep learning, and natural language processing. General AI, or strong AI, refers to a more ambitious goal: creating machines that possess the ability to understand, learn, and apply knowledge in a way that is indistinguishable from human intelligence. While general AI remains a theoretical aspiration, the progress in narrow AI continues to drive substantial advancements across various domains.

In recent years, advancements in computational power, the availability of vast amounts of data, and sophisticated algorithms have propelled AI to new heights. Machine learning has enabled computers to achieve and even surpass human performance in various tasks such as image and speech recognition, strategic game playing, and language translation (LeCun, Bengio & Hinton (2015)). AI's impact is evident in everyday applications like virtual assistants, and recommendation systems, and autonomous vehicles, revolutionizing how we interact with technology (Goodfellow, Bengio & Courville (2016), Stilgoe (2017)). In healthcare, AI-powered systems assist in diagnosing diseases, personalizing treatment plans, and predicting patient outcomes (Topol (2019)). In finance, AI algorithms enhance fraud detection, algorithmic trading, and risk management (Trivedi, Bhagchandani & others (2018)). Moreover, AI is revolutionizing industries like manufacturing, retail, entertainment, and education, fostering efficiency and innovation.

1.2 Machine Learning

Machine Learning (ML) is a dynamic and rapidly evolving field within AI that focuses on developing algorithms and statistical models enabling computers to perform tasks without explicit instructions. Instead of following predetermined rules, machine learning systems learn from data, identifying patterns, and making decisions with minimal human intervention.

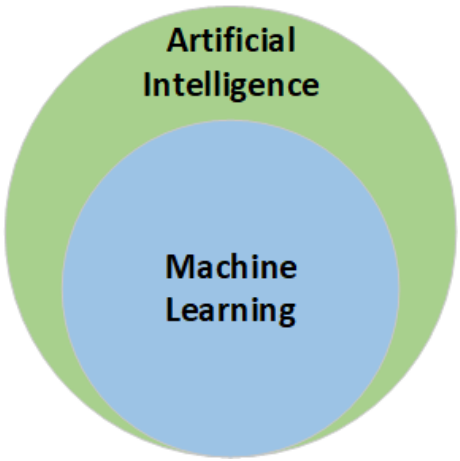

Figure 1.1 Artificial Intelligence vs Machine Learning

The heart of machine learning is the concept of training models using data. These models can be categorized into three primary types: supervised learning, unsuper-

vised learning, and reinforcement learning. Supervised learning involves training a model on a labeled dataset, where the desired output is known, allowing the model to learn the relationship between input features and the target outcome. Common applications of supervised learning include image classification, speech recognition, and medical diagnosis (Michalski, Carbonell & Mitchell (2013)).

Unsupervised learning, on the other hand, deals with unlabeled data, where the model must discern patterns and structures from the input data without explicit guidance. Techniques such as clustering and dimension reduction fall under this category, with applications ranging from customer segmentation in marketing to anomaly detection in network security (Murphy (2012)).

Reinforcement learning involves training an agent to make a sequence of decisions by rewarding it for desirable actions and penalizing it for undesirable ones. This type of learning is particularly prominent in robotics, gaming, and autonomous systems, where agents learn optimal strategies through trial and error (Sutton & Barto (2018)).

Training model involves a learning algorithm that iteratively adjusts the model's internal parameters to minimize prediction errors. The term "model" can thus encompass several levels of specificity, ranging from a broad class of models and their corresponding learning algorithms to a fully trained model with fine-tuned internal parameters (Russell et al. (2010)).

In the realm of machine learning, various types of models have been explored and utilized. The process of choosing the most suitable model for a particular task is known as model selection. Some of the prominent examples of ML models include Artificial Neural Networks, Logistic Regression, Support Vector Machines, Naive Bayes, Decision Trees, Linear Regression, and Random Forests. Each of these models has unique characteristics and is chosen based on the specific requirements and nature of the task at hand.

The advent of deep learning, a subset of machine learning that utilizes neural networks with multiple layers, has led to remarkable breakthroughs in the field. Deep learning models, inspired by the human brain's architecture, excel at handling vast amounts of unstructured data, such as images, audio, and text. This has enabled significant advancements in computer vision, natural language processing, and speech synthesis (LeCun et al. (2015)).

1.3 Deep Neural Network

DNNs have emerged as a pivotal technology in the field of AI, revolutionizing the way machines perceive, learn, and make decisions. Originating from the broader concept of artificial neural networks, as depicted in Figure 1.3, DNNs are characterized by

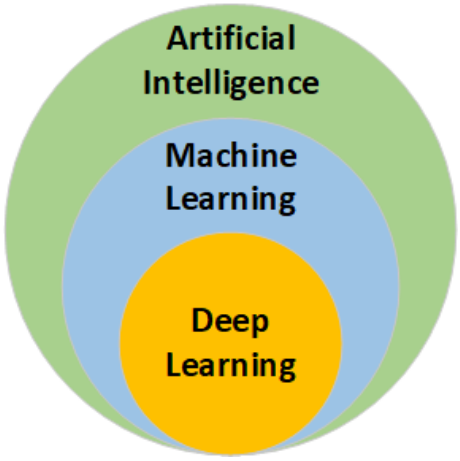

Figure 1.2 Artificial Intelligence vs Machine Learning vs Deep Learning

their multiple layers of interconnected neurons that mimic the structure and function of the human brain. These networks are capable of automatically learning hierarchical representations of data, which makes them exceptionally powerful for a wide range of complex tasks.

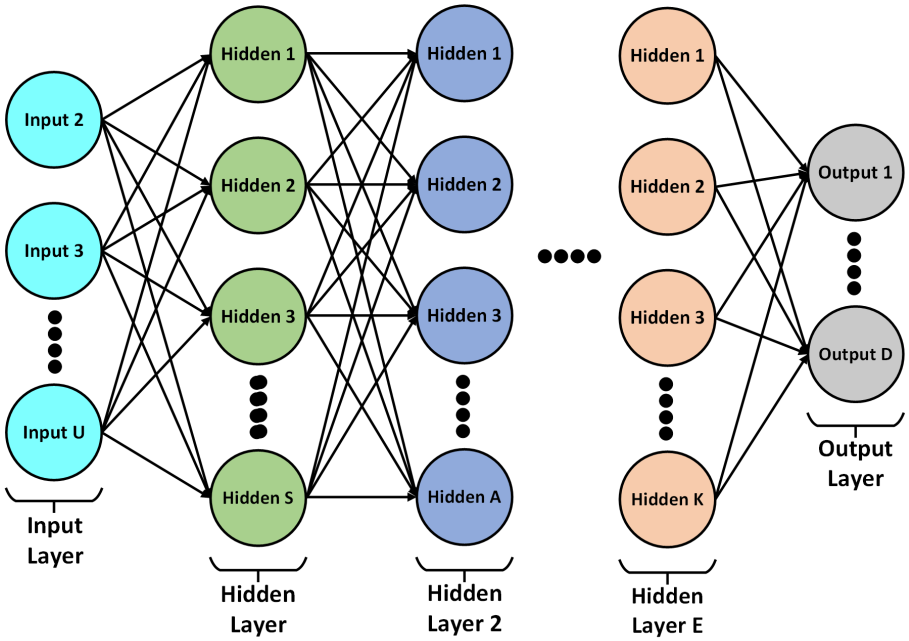

Figure 1.3 General DNN Architecture

The resurgence of interest in DNNs can be attributed to several key factors. First, the exponential growth in data generation across various domains has provided the necessary fuel for training these models. Second, advancements in computational power, particularly with the advent of Graphics Processing Units (GPU) and specialized hardware like Tensor Processing Units (TPU), have enabled the efficient training of deep networks. Third, the development of sophisticated algorithms and techniques, such as back propagation, dropout, and various optimization methods, has enhanced the performance and generalization capabilities of DNNs (LeCun et al. (2015)).

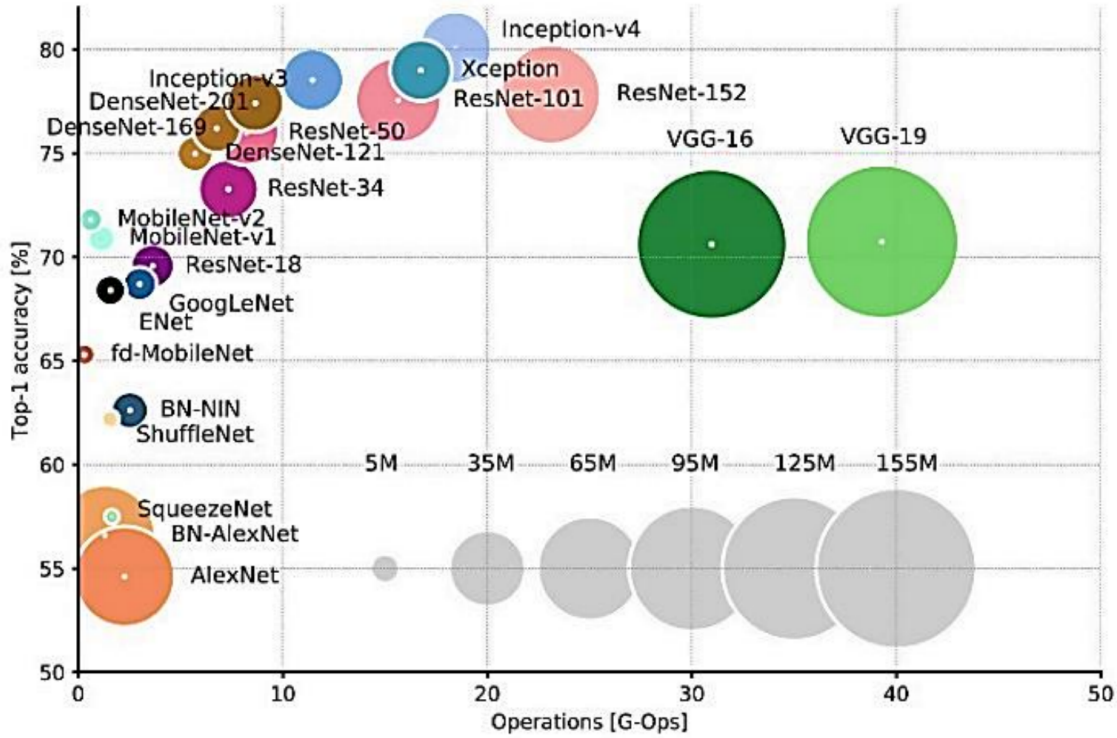

Figure 1.4 Various Models (Fauzi et al., 2021)

DNNs have demonstrated remarkable success in numerous applications, including image and speech recognition, natural language processing, and game playing. For instance, DeepMind AlphaGo has 89 layers (Silver, Huang & others (2016)), and ResNet model, which is used for image classification, can have a maximum of 152 layers (He, Zhang & others (2015)). This makes the neural networks capable of learning more complex and abstract features, therefore, improving the classification accuracy. Convolutional neural networks (CNN), a type of DNN, have achieved state-of-the-art results in image classification and object detection tasks (Krizhevsky, Sutskever & Hinton (2012b)). Similarly, recurrent neural networks (RNN) and their variants, such as Long Short-Term Memory (LSTM) networks, have shown superior

performance in sequential data processing tasks, such as language translation and time series prediction (Hochreiter & Schmidhuber (1997)).

As can be seen at Figure 1.4, particularly image related model parameter size goes about million to 100 millions. Most of the parameters are contributed by the fully connected layers.

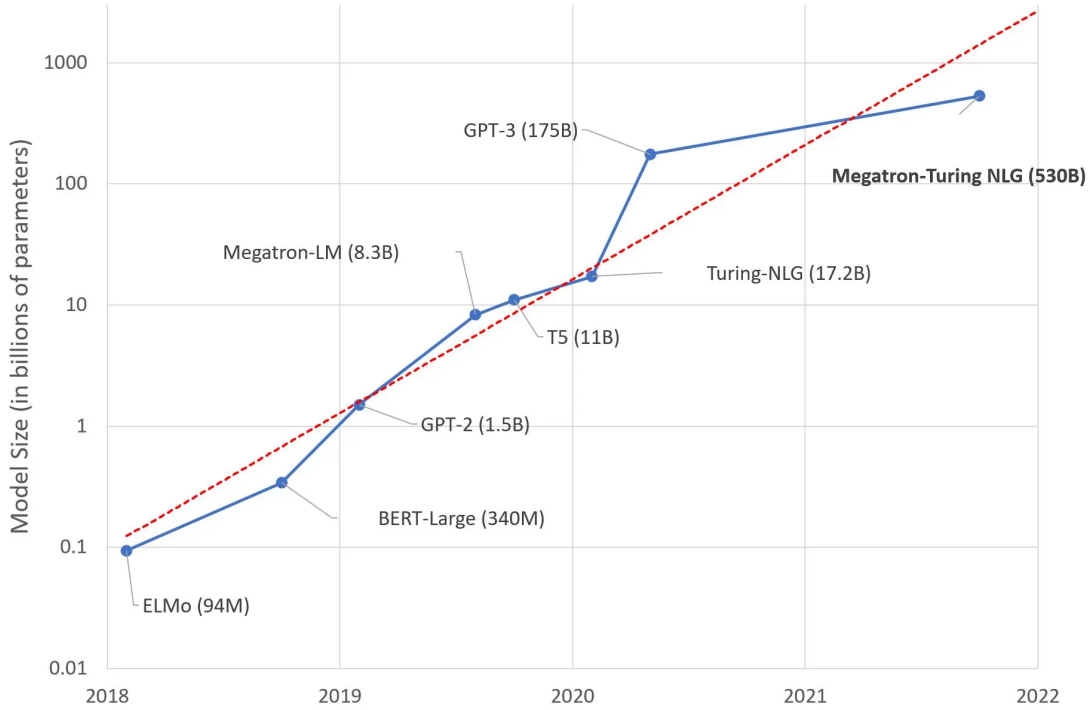

Figure 1.5 Various Large Language Models (LLM) (Mehra, 2023)

In today, Large Language Models (LLM) are a significant leap forward in the field of artificial intelligence and natural language processing. Their ability to understand and generate human language with remarkable accuracy has paved the way for a wide range of applications, transforming industries and enhancing our interactions with technology (Zhao & others (2023)). As shown in Figure 1.5, LLM's parameter reaches up to billions.

These AI models are increasingly integrated into everyday life, expanding their presence across various devices such as computers, mobile phones, and Internet of Things (IoT) devices. IoT devices and their DNN applications are explored in the next section.

1.4 Internet of Things

The Internet of Things (IoT) refers to a network of physical objects (devices), vehicles, appliances, and more that use sensors, software, and other technologies to connect and exchange data with other devices and systems over the internet. These "smart" devices collect and share data to improve efficiency, reduce human intervention, and enhance user experience (Greengard (2021)).

Figure 1.6 Internet of Things (Muha, 2019)

IoT devices consist of key components such as sensors and actuators, connectivity options, data processing capabilities, and user interfaces. Sensors gather data from the environment (e.g., temperature, humidity, motion), while actuators perform actions based on processed data (e.g., turning on lights, adjusting thermostats). Connectivity is achieved through various means like Wi-Fi, Bluetooth, cellular networks, or Zigbee, allowing devices to communicate and exchange data. Data collected by sensors is processed either locally on the device itself or remotely in the cloud. User interfaces, such as mobile apps or web dashboards, enable users to monitor and control these devices (Rose & Eldridge (2015)).

IoT technology is applied in various domains, including smart homes, healthcare, industrial IoT, smart cities, and agriculture. In smart homes, devices like smart thermostats, lighting systems, and security cameras provide convenience, security, and energy efficiency. In healthcare, wearable devices monitor vital signs, track fitness, and manage chronic diseases, providing real-time data to healthcare providers.

Industrial IoT is used in manufacturing for predictive maintenance, improving operational efficiency, and automating processes. Smart cities benefit from IoT through smart traffic management, waste management, and energy-efficient buildings. In agriculture, IoT sensors monitor soil conditions, weather, and crop health, optimizing irrigation and increasing yields (Gubbi, Buyya & others (2012)).

The benefits of IoT include improved efficiency and productivity, enhanced decision-making with real-time data, increased automation, and better resource management and sustainability. However, challenges such as security concerns, privacy issues, interoperability between different devices and standards, and managing the vast amount of data generated by IoT devices need to be addressed. Security vulnerabilities in devices and privacy issues related to data collection and use are significant concerns (Atzori, Iera & Morabito (2010)).

IoT devices are deployed for the various DNN applications such as face recognition, human activity recognition and activity tracking (Kodali, Hansen & others (2017)). Running a DNN on the cloud introduces a significant dependency on wireless connectivity. In many areas, wireless connections may be nonexistent, poorly established, or slow. This poses a serious challenge for high-risk and real-time applications, such as driverless cars, where continuous, reliable, and real-time DNN processing services are essential for ensuring safety and security (Zhang & Kouzani (2020)). To reduce the internet bandwidth usage, the latency and increase the security, edge computing is the one of the promising solution. Edge computing brings another problem for the IoT devices since running DNN requires significant energy consumption for the Multiply and Accumulate (MAC) operations for the computational units using digital DNN hardware, which is the case mostly. Especially when the model gets larger, parameter size increases and that requires frequent memory access. Different hardware types are examined in the next section for the IoT devices.

1.5 Hardware Comparison

In literature there are some survey and comparison of hardware types for AI applications in terms of speed, power, flexibility and accuracy (Shahid & Mushtaq (2020), Amanatidis, Iosifidis & Karampatzakis (2022), Guo, Zeng & others (2018)). Each type of hardware has its own advantages and disadvantages for AI applications.

1.5.1 Central Processing Unit

The Central Processing Unit (CPU) is the primary component of a computer that performs most of the processing. It carries out instructions from programs by performing basic arithmetic, logic, control, and input/output (I/O) operations. The CPU is often referred to as the "brain" of the computer because it is essential for executing programs and managing the computer's operations.

The performance of a CPU is determined by several factors, including its clock speed (measured in GHz), the number of cores it has (which allows for parallel processing), and its architecture. Modern CPUs are often multi-core, meaning they contain multiple processing units within a single chip, which can significantly enhance performance for tasks that can be parallelized.

Main disadvantage of running DNN model on CPU are slow response of the model and high energy consumption since CPUs are not optimized for the DNN operations that require massive matrix multiplications. Figure 1.4 shows that moderate model has about millions of parameters. Multi-core CPU provides parallelism up to certain point however it is not enough for real-time applications especially when the latency becomes extremely important. Instruction fetch, decode and going back and forth to memory result in a considerable overhead for the DNN applications.

On the other hand, it is easy to deploy a CPU for small models in IoT device. Software implementation requires less time. CPU is flexible so that most of the models can be implemented and if necessary, updating or changing the model is easy. It can handle floating point operations so activation functions such as Sigmoid and Hyperbolic Tangent can be easily processed in CPU.

1.5.2 Graphics Processing Unit

The Graphics Processing Unit (GPU) is a specialized electronic circuit designed to accelerate the rendering of images, videos, and animations. It is particularly effective at handling the complex mathematical calculations required for rendering graphics. Originally developed to render graphics for computer games, GPUs are now used in a wide range of applications beyond gaming, including scientific computing, artificial intelligence, and cryptocurrency mining.

GPUs are designed with many smaller cores that can handle multiple tasks simul-

taneously. Unlike CPUs, which are optimized for sequential processing, GPUs are optimized for throughput and parallelism. This means they can process many operations concurrently, making them ideal for workloads that can be divided into smaller, parallel tasks such as rendering graphics or performing large-scale computations (Owens & others (2008)). GPUs typically have high memory bandwidth to quickly move large amounts of data in and out of the GPU for processing. This is crucial for handling the data-intensive nature of graphic rendering and scientific computations.

Compute Unified Device Architecture (CUDA) cores are parallel processors within NVIDIA GPUs, designed for efficient computation across a variety of tasks, especially scientific, engineering, and deep learning applications (Nickolls & Dally (2010)). A CUDA core is a single processing unit within an NVIDIA GPU, similar to a CPU core but optimized for handling parallel workloads. GPUs typically house thousands of CUDA cores, enabling simultaneous execution of many operations, contrasting with the fewer but more powerful cores in CPUs optimized for serial processing. These CUDA cores are integral components of Streaming Multiprocessors within the GPU, which also include warp schedulers and memory.

In the context of DNN, CUDA cores are particularly valuable due to their ability to parallelize the highly computational tasks involved in training and inference. DNNs require extensive matrix multiplications and additions, which CUDA cores can efficiently perform in parallel, thereby accelerating the entire process. During the training phase, CUDA cores facilitate the rapid computation of forward passes (propagation of data through network layers) and backward passes (calculation of gradients for backpropagation) (Zhang, Gunupudi & Zhang (2015)). For inference, CUDA cores enable fast execution, crucial for real-time applications such as image recognition and natural language processing (Dtv & Ramana (2019)).

GPUs are suitable for the running DNN models especially by utilizing the CUDA cores. GPUs can enable real-time processing and decision-making, which is crucial for many IoT applications such as autonomous vehicles, surveillance, and industrial automation. However it consumes large amount of power that creates problem for battery powered devices. On the other hand, GPUs are not small enough and heat dissipation is another problem to make a compact IoT device.

1.5.3 Field-Programmable Gate Array

An FPGA, or Field-Programmable Gate Array, is a type of semiconductor device that can be configured by the user after manufacturing. Unlike traditional processors (CPUs and GPUs), which have fixed architectures, FPGAs consist of an array of programmable logic blocks and interconnects that can be reconfigured to perform specific tasks.

FPGAs can be reprogrammed to execute different tasks, making them highly versatile. They support massive parallelism by allowing many operations to be executed simultaneously. Users can design custom hardware tailored to specific applications, optimizing performance and efficiency. FPGAs can provide low-latency processing because they can be configured to execute tasks in hardware, rather than software.

In DNN context, FPGAs can be programmed to create custom accelerators for various stages of DNN processing, such as convolution, pooling, and fully connected layers. This allows for optimization specific to the neural network architecture and workload. FPGAs are often used for accelerating inference in DNN (Tsai, Ho & Sheu (2019))s. Their ability to execute highly parallel tasks with low latency makes them suitable for applications requiring real-time processing, such as autonomous driving, robotics, and video analytics. Although more commonly used for inference, FPGAs can also be configured to accelerate the training of neural networks. Custom architectures can be designed to handle the massive matrix multiplications and other operations involved in training. FPGAs can be more energy-efficient than GPUs for certain tasks because they can be optimized for specific operations, reducing unnecessary power consumption. The reconfigurability of FPGAs makes them ideal for prototyping and deploying DNNs, especially when the architecture may change or evolve over time. This flexibility is beneficial in research and development environments where algorithms are continually refined (Shawahna, Sait & El-Maleh (2018)). In edge computing scenarios, FPGAs are deployed to run DNN inference close to the data source (e.g., cameras, sensors), reducing the need for data transfer to central servers and thereby minimizing latency and bandwidth usage (Suda, Chandra & others (2016)). An FPGA-based scalable Deep Learning Accelerator Unit (DLAU) has been proposed, featuring three pipeline processing units to enhance throughput. This design employs tiling techniques, FIFO buffers, and pipelines to maximize the reuse of computing units and minimize memory transfer operations (Wang, Yu & others (2016)).

1.5.4 Application-Specific Integrated Circuit

Application-Specific Integrated Circuit (ASIC) is a type of integrated circuit (IC) that is custom-designed for a specific application, rather than being intended for general-purpose use. ASICs are optimized to perform a particular set of tasks very efficiently, making them highly effective for specific applications. Due to their specialized nature, ASICs can be more energy-efficient than other hardware types because they do not include unnecessary components and can be optimized for power consumption. It can be designed to occupy less space on a chip that can be advantageous especially in compact devices.

In DNN applications context, ASICs are commonly used to accelerate the inference phase of DNNs. They are designed to handle specific neural network operations with maximum efficiency, providing fast and low-latency inference. ASICs are ideal for edge devices where power efficiency and compactness are critical. They enable real-time inference directly on devices such as smartphones, IoT devices, and autonomous systems without relying on cloud-based processing. ASICs support custom neural network architectures and operations, ensuring that the hardware is perfectly suited to their specific DNN workloads. At architecture level, some of the accelerators have been designed to handle neural network tasks.

Google's Tensor Processing Units (TPU) are a prime example of ASICs designed specifically for accelerating machine learning workloads (Jouppi, Young & others (2017)). The Eyeriss accelerator, as proposed, enhances energy efficiency and throughput by implementing a new processing dataflow (Chen, Krishna, Emer & Sze (2017)). This approach reduces costly data movement by maximizing local data reuse.

As a result, ASIC based DNN accelerators are the most suitable hardware for the IoT devices in terms of power consumption, performance and compactness.

1.5.5 Overall Comparison

There are lots of commercially available CPU, GPU and FPGA along with different specialization, manufactured at different technology node. Ignoring all the differences, generally the Table 1.1 is valid for the hardware comparison.

Hardware	Flexibility	Power Consumption	Latency
CPU	Highest	Moderate	High
GPU	High	High	Moderate
FPGA	Moderate	Low	Low
ASIC	Low	Lowest	Low

Table 1.1 Hardware Comparison

1.6 Motivation

In recent years, usage of AI has increased in edge computing and processing applications (López, Montresor & others (2015)). The reasons are the enormous amount of data being collected, increasing in computational power and high data traffic (Wang et al. (2020)) as depicted in Figure 1.7. Moreover, IoT applications also have been popular and there is a high interest in using IoT devices with AI. To be able to run a DNN application on IoT devices, these devices need to be energy-efficient and need to have the edge-computing capability since they are mostly not connected to a continuous power supply, they have poor wireless internet connection or not at all.

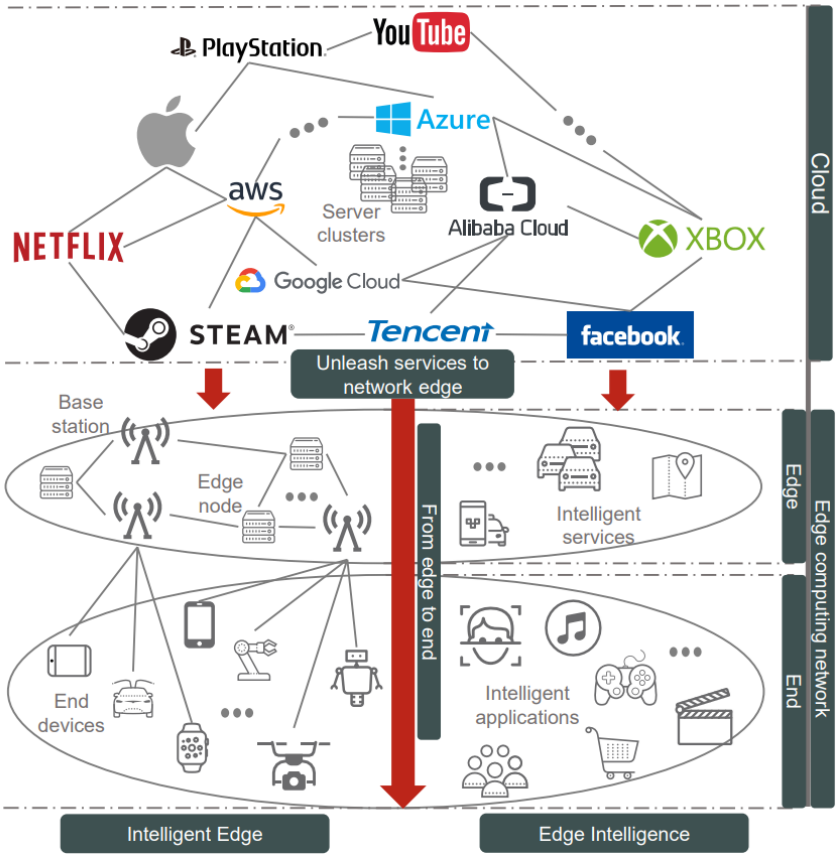

Figure 1.7 Edge Intelligence and Intelligent Edge (Wang et al., 2020)

Edge computing eliminates the necessity of sending data to cloud, thus workload on internet traffic is reduced. At the same time, latency is reduced because data is not being sent to a remote location to be processed and response of the remote location is not waited by the edge device. Additionally, security of edge device and the data itself increases since device is not connected to a public network thus edge devices are protected from remote cyber-attacks (Chen & Ran (2019)). According to a study, nearly half of IoT applications require edge computing capability for latency, bandwidth and security reasons (Srivatsa (2018)). A quick comparison is shown in Figure 1.8.

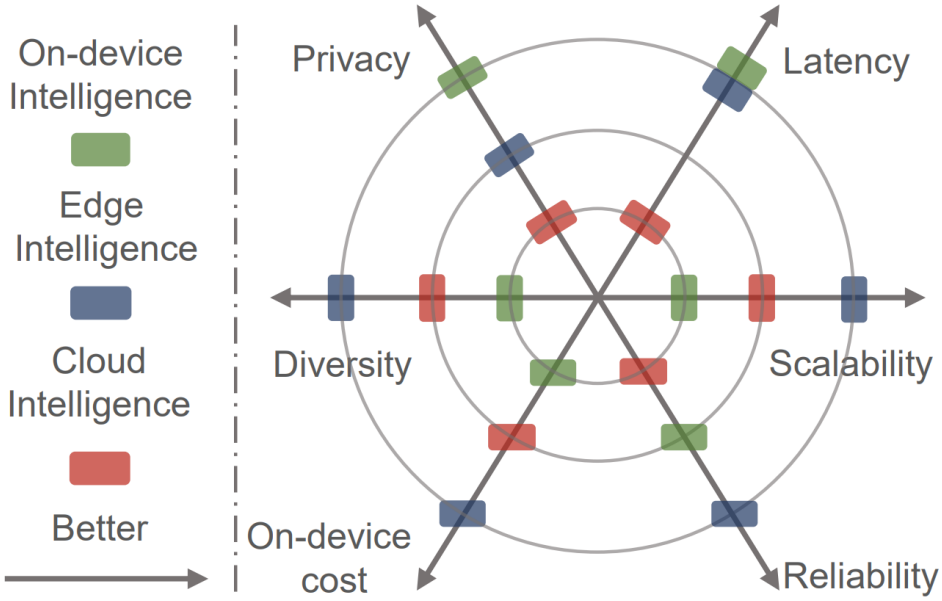

Figure 1.8 Capabilities comparison of cloud, on-device and edge intelligence (Wang et al., 2020)

DNN applications require large number of matrix multiplications. Low-cost and energy efficient chips are needed to operate DNN applications on edge devices. Voltage underscaling is one of the used method to reduce power consumption since power consumption is proportional to the operating voltage square. With the same operating clock frequency, timing errors arise while reducing the supply voltage and impact of these bit errors may or may not be tolerable by the DNN model itself.

Process, voltage and temperature (PVT) variations affect the timing error probability. Change in these effects may speed up the circuit or slow down. Therefore, taking into account PVT variations is a necessity in digital integrated circuit that operates with a small margin for timing errors. Conventional method that gate-level simulation takes huge amount of time to observe the effect of the PVT variations on

the circuit and eventually on DNN model in terms of inference accuracy. In DNNs, several optimizations can be implemented at the hardware level, including altering the circuit architecture or modifying the number of bits (Jouppi et al. (2017), Chen et al. (2017), Sharma, Park & others (2016)). Each minor adjustment necessitates validation, leading to an increased design and verification duration for the final hardware system. To be able to make the design faster, a crosslayer platform is desired to be developed as depicted in Figure 1.9. It takes inputs as process variation, supply voltage variation, temperature variation, netlist of the hardware and operating conditions (supply voltage, temperature, frequency).

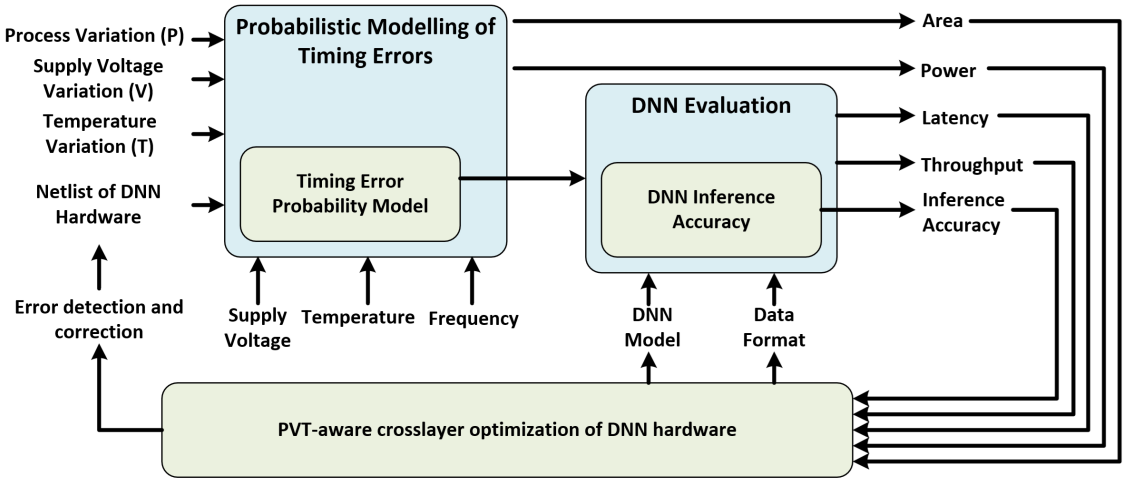

Figure 1.9 Crosslayer Platform for Analyzing Hardware

The primary objective of this thesis work is to develop a PVT aware probabilistic timing error model, which allows for efficient analysis of the provided netlist for the crosslayer platform. This ultimately contributes to the development of rapid DNN hardware for highly energy-efficient IoT applications. Moreover as shown in Figure 1.9, this model considers power supply noise that might be the result of high switching activities on the gates (Enami, Ninomiya & Hashimoto (2009)) or external effects such as electromagnetic interference (Moon, Kim & others (2015)), temperature variation due to workload of the chip (Iranfar, Terraneo & others (2017)). Designed DNN hardware (gate level) could be analyzed by utilizing the error probability model without performing time consuming simulations to observe error probability on the bits. These bit errors are injected to DNN layers by the platform. Thus, designer observe the effect of PVT variations on the inference accuracy quickly and depending on the analysis result, designer can make changes if necessary and apply dynamic voltage scaling to reduce power consumption according to importance of the DNN's layer.

1.7 Thesis Organization

The next chapter focuses on the probabilistic timing error model. By taking into account Process, Voltage, Temperature (PVT) variations in the case of supply voltage reduction for reducing the energy consumption, bit error probability should be evaluated find to observe the effect of errors on the DNN model, eventually inference accuracy. Probabilistic model is developed to find error probability of the bits without relying on the time consuming Monte Carlo (MC) simulations. That makes the analysis faster with decent accuracy.

In chapter 3, DNN accelerator implementation is examined, how it works and digital blocks are explained. DNN calculations requires multiply and accumulate (MAC) operation extensively. Systolic MAC array architecture is chosen for accelerator implementation. It consists of 16x16 MAC array, means accelerator supports 16 fully connected neurons for a single acceleration with 16 input combinations. The accelerator works at 200 Mhz clock frequency at nominal voltage $1.2V$. Designed chip manufactured in 65nm CMOS technology.

The final chapter discusses the summary of the work undertaken and explains the future work to be completed.

2. Probabilistic Timing Error Model

2.1 Introduction

In the literature, most of the studies on improving the energy efficiency of DNNs are based on network structure (Han, Mao & Dally (2016), Judd, Delmas, Sharify & Moshovos (2017)), circuit architecture (Jouppi, Borchers & others (2017), (Sharma et al. (2016))) and changing the data type or reducing number of bits (Courbariaux, Hubara & others (2016), Vanhoucke, Senior & Mao (2011)). In addition, recently developed memory technologies that enable in-memory computing are also important areas of research, as they reduce the energy consumption by eliminating the reading data from memory and writing data back to memory (Yan, Cherian & others (2020), Halawani, Mohammad & others (2019)).

One of the most popular ways to increase energy efficiency in digital hardware is to reduce the supply voltage. However, this affects the operating performance of the circuit (slowing down) and increases the probability of timing errors in the circuit. The timing error probability is directly proportional to how much the supply voltage is reduced from the nominal operating level. On the other hand, as in all semiconductor circuits, the performance of the circuit may change due to process and temperature variations. Especially when the supply voltage is reduced, due to process and temperature variation, the errors increase because of the combination of all negative factors. Therefore, when developing energy-efficient digital hardware for DNN accelerators, it is necessary to take into consideration process variation, supply voltage and temperature (PVT) in order to properly analyze how lowering the supply voltage affects the timings and eventually inference accuracy.

Generally, gate-level simulations are performed to verify the digital circuit. The problem is that these standard cell libraries provided by the foundry are character-

ized at certain voltage levels. If a designer wants to see the timing on the digital circuit at different voltage level apart from the provided by the foundry, conventionally there are two options. First one, designer has to create their own library at desired voltage level to be operated, for Electronic Design Automation (EDA) tools such as Design Compiler or Innovus for timing analysis. Thus, designers can perform timing analysis easily. However, library characterization requires additional tool. Even in this scenario, time-consuming gate level simulations are required.

Second one, designer has to perform transistor level simulation at desired voltage level to be operated which is time consuming more than gate-level simulations.

Modern DNN models consist of hundreds of layers. Millions of parameters are used and high number of operations are performed. Depending on the DNN model to be accelerated, voltage reduction might have huge impact on some layers, little effect for other layers. Dynamic voltage scaling is one of the possible solution.

DNN accelerators consist of a large number of circuit elements and gate-level simulations take a lot of time. Simulation time increases proportional to the complexity of the DNN model used, and the number of gates used in the integrated circuit. As mentioned above, every situation needs to be simulated after the slightest change in the integrated circuit. Simulations significantly increase design time. The size and complexity of modern DNN, as well as the ability to make changes to many parameters to save energy as mentioned above, increase the importance of simulation time. During the design phase, being able to quickly learn how a change in each parameter affects inference accuracy, will provide a significant advantage for the designer.

For that reason, it is necessary to be able to calculate accurately the timing errors taking into account PVT variations when the supply voltage is reduced. As a result, high accuracy timing model which takes all factors into account, is required to accelerate development process of the digital circuit for DNN applications without relying on the time-consuming transistor level simulations. The following section explains how the probabilistic timing error model works.

2.2 Probabilistic Error Calculation

DNN accelerators such as the systolic MAC array architecture, has high amount of parallel synchronous components. In highly parallel synchronous circuits, the concurrent switching of many registers generates a significant current drawn from the power networks, leading to voltage fluctuations at the power lines (Wang & Salman (2017)). In compact devices, where many circuit components are densely placed on the Printed Circuit Board (PCB), electromagnetic interference can lead to fluctuations in the power supply (Moon et al. (2015)).

The variations on the power supply voltage can be described as a random variable following a Gaussian distribution. The standard deviation, denoted as (σ), represents the power supply noise, while the mean, denoted as (μ), corresponds to the nominal supply voltage (Enami et al. (2009)). Given the distinct voltage-delay characteristics of the technology, the timing probability distribution can be modeled based on the specified voltage distribution. The resulting probability density function (PDF) is utilized in the proposed modeling approach to calculate the timing error probability.

In the literature, probabilistic timing analysis has been done by only considering the supply voltage level and power supply noise (Rathore et al. (2020)). This work is extended by integrating temperature and process variations. Figure 2.1 shows the steps to calculate the probabilistic timing error at any node in a digital circuit.

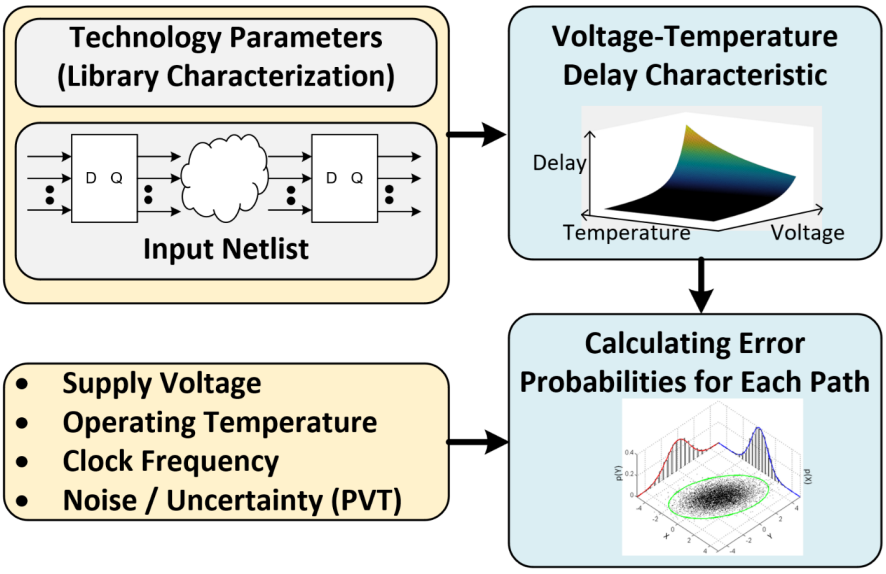

Figure 2.1 Key steps in the proposed approach to analyze timing error probabilities

For the probabilistic error calculation, standard gates are characterized for different voltage-temperature pairs. Additionally, standard deviation due to process variation is extracted for each gate. Characterization methodology is explained in section 2.3. Then extracted gate delays are used for the calculation of the path delay similar to Static Timing Analysis (STA). Standard deviation due to process variation is added to analysis that explained in section 2.4.6.

Initially voltage variation aware model (V Model) which was proposed by the Rathore and others (Rathore et al. (2020)), is demonstrated for 65nm CMOS technology. Then temperature variation is integrated to voltage aware model and voltage-temperature aware model is obtained (VT Model). Lastly, process variation is added to VT Model and process-voltage-temperature aware model (PVT Model) is developed. Timing error probability of the given netlist is calculated and compared with 2 different methodology.

- First one is Monte Carlo (MC) simulations and this result is taken as reference for the finding error probability. It is referred as **"MC Simulation"** in the comparison tables. Process, voltage and temperature are varied separately or together with given nominal (μ) value and sigma (σ) value. Then error probability is calculated based on the number of failed simulations over total number of simulations.

- The second method calculates the error probability by integrating the PDF of the variables which explained in the section 2.4. It is referred as **"P,V,T Model STA"** in the comparison tables. Voltage and temperature characteristic of the given path is calculated by utilizing extracted gate delays similar to STA tools. Each letter corresponds to an effect that model considers to calculate error probability. P, V, T represent process, voltage, temperature respectively.

2.3 Gate Characterization

In the proposed model, gates that have been used in the design of MAC Unit, were characterized in a simpler way to find path delay without simulating the circuit. Thus path delay is calculated in a faster way to analyze error probability. High to low propagation delay (τ_{PHL}) and low to high propagation delay (τ_{PLH}) are defined as shown in Figure 2.2. Delay of the gate is defined as their average as shown in Equation 2.1.

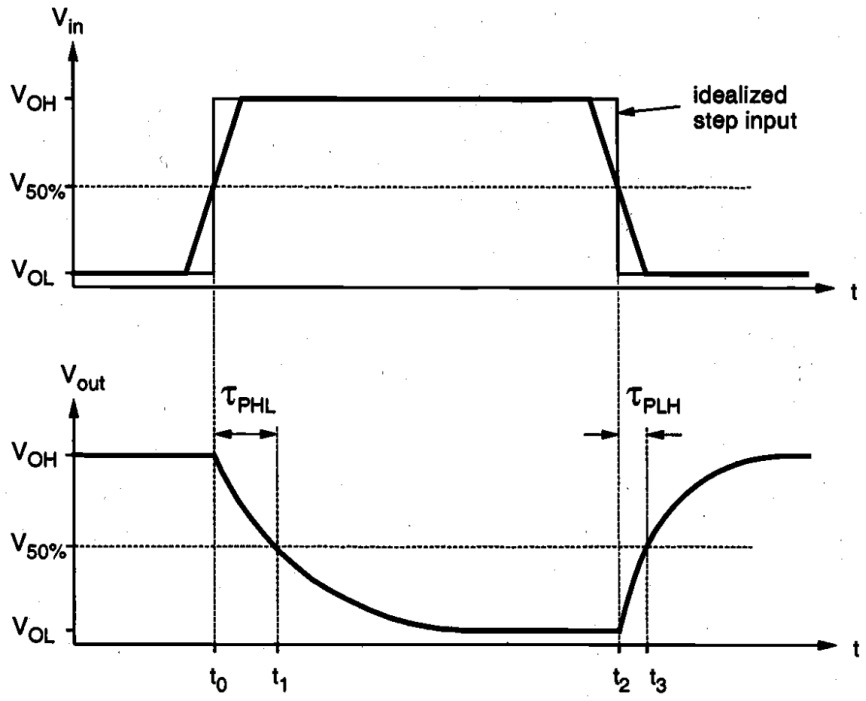

Figure 2.2 Propagation Delay Definition (Kang & Leblebici, 2003)

$$\tau_{PHL} = t_1 - t_0$$
$$\tau_{PLH} = t_3 - t_2$$

$$\tau_p = \frac{\tau_{PHL} + \tau_{PLH}}{2} \tag{2.1}$$

Example setup circuit of single input gate and multiple input gate are shown in Figure 2.3 and 2.4 respectively. Ideal source creates pulse signal ranging between 0 to VDD with $50ps$ rise and fall times suggested by the foundry. Since ideal source is capable of driving gates regardless of the Fan-out or input capacitance of the gate to be characterized, weakest buffer is used in between ideal source and gate to be characterized. Also, another buffer connected as a load. This is not a full characterization since it represents only one case. Driving cell of the input, Fan-out capacitance etc. can change depending on the design. However, demonstrated setup enables a quick analysis for design. For the multiple input gates, pulse was applied to only one input that results transition on the output and other inputs remain the same.

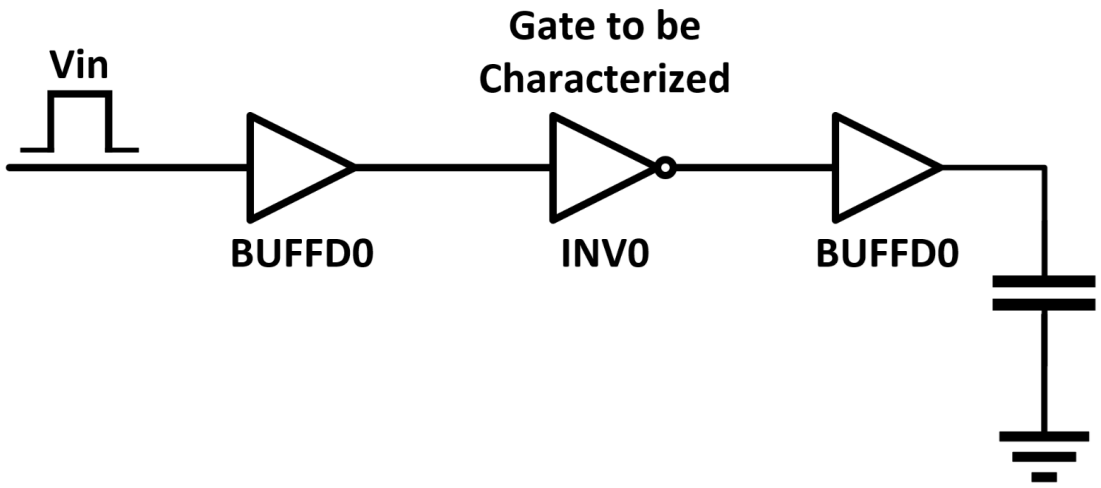

Figure 2.3 INVD0 Characterization Setup

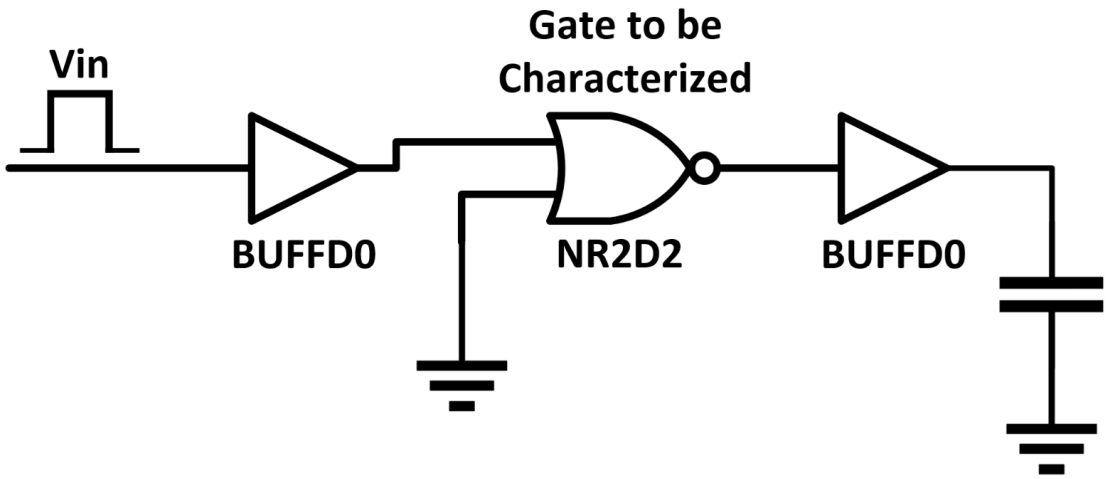

Figure 2.4 NR2D2 Characterization Setup

For each gate, voltage and temperature were swept between $0.6V$ to $1.2V$ with $5mV$ steps, $-20°C$ to $100°C$ with $10°C$ steps respectively. Thus $13x121$ temperature, voltage lookup table was obtained. Then 2D interpolation was used to increase the resolution to $1mV$ and $1°C$. This lookup table was used for the calculation of the path delay.

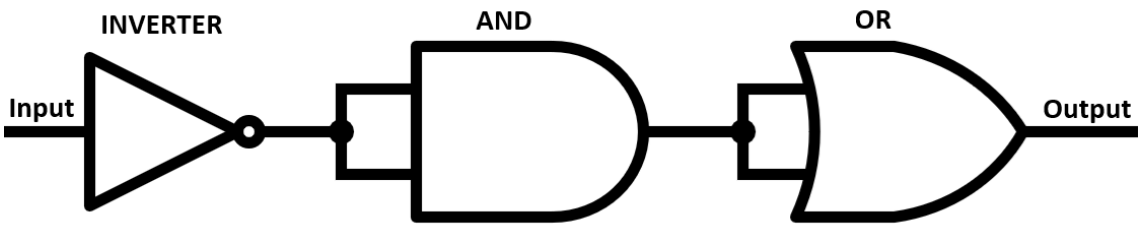

Figure 2.5 Simple Combinational Circuit

An example of simple combinational circuit is depicted in Figure 2.5. It's path delay calculation is shown in Figure 2.6.

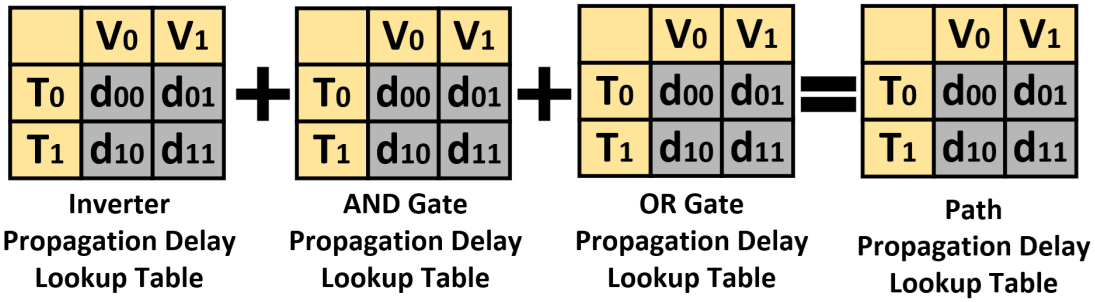

Figure 2.6 Path Delay Calculation

Calculated path delay is used for further error probability calculation for the path.

2.4 Probabilistic Timing Error Models

2.4.1 Timing Probability Distribution Considering Power Supply Noise

Voltage supply level determines the timing delay of the path. For the demonstration of the analysis, an example circuit shown in Figure 2.7 is used. Initially voltage and delay characteristic is extracted by performing simulation at different voltages which is given in Figure 2.8.

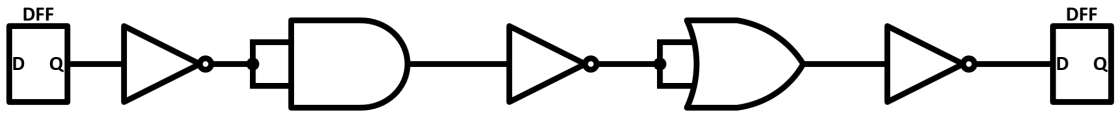

Figure 2.7 Sequential Circuit

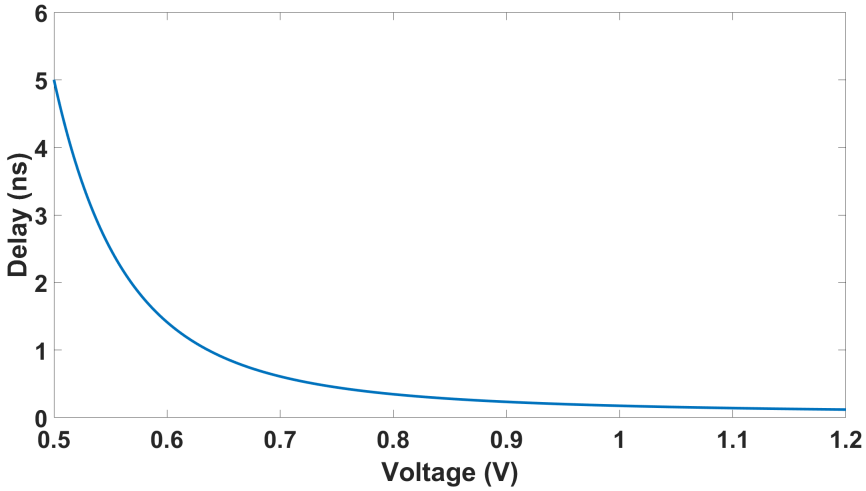

Figure 2.8 Voltage versus Propagation Delay

As it can be seen from the voltage and delay characteristic of the example circuit, when voltage increases, delay decreases. Then, VDD is varied as Gaussian distribution with the introducing of the noise (σ) as previously explained. Probability Density Function (PDF) of the voltage is shown on Figure 2.9 for different noise value. Magnitude of the supply noise can vary depends on the internal switching and environmental effects. The noise level is chosen based on the referenced study (Rathore et al. (2020)).

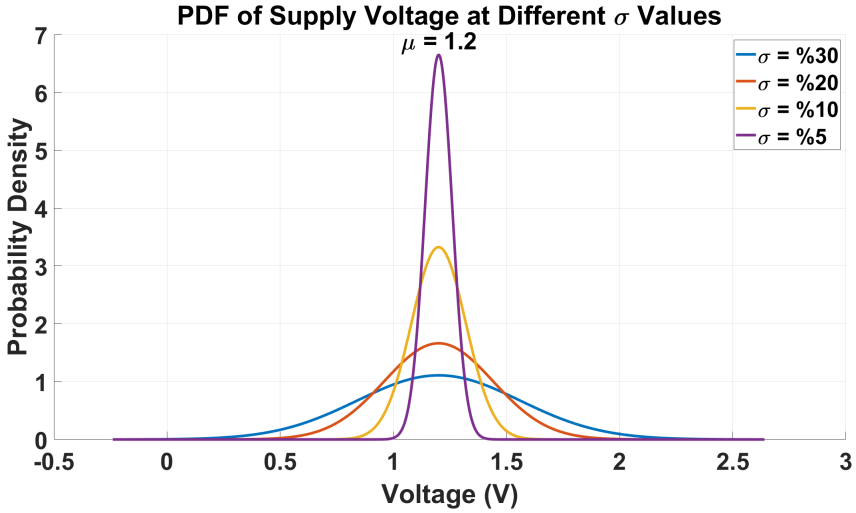

Figure 2.9 Voltage PDF

Since voltage and timing delay has one to one relation, the timing probability distribution is found from the voltage distribution by applying the change of variable technique (Ross (2010)). According to change of variable technique, if X is a continuous random variable with the PDF f_X, then the probability density function of

the random variable Y, defined as Y = g(X) as given in Equation 2.2.

$$f_Y(y) = f_X[g^{-1}(y)]\left|\frac{d}{d_y}g^{-1}(y)\right|. \tag{2.2}$$

Timing delay of the path can be represented as function of voltage $v(t)$ at time t. Given the voltage PDF $f_V(v)$ with the random variable of voltage (v), corresponding timing PDF can be found as a function of the timing random variable $v(t)$. The timing PDF $f_T(t)$ can be determined as shown in Equation 2.3,

$$f_T(t) = f_V[v(t)]\left|\frac{d}{d_t}v(t)\right|. \tag{2.3}$$

The voltage PDF of example circuit in given Figure 2.9 corresponds to, timing PDF of the example circuit in Figure 2.10. Asymmetric delay distribution is result of nonlinear voltage-delay relationship as shown on Figure 2.8, especially at lower voltages.

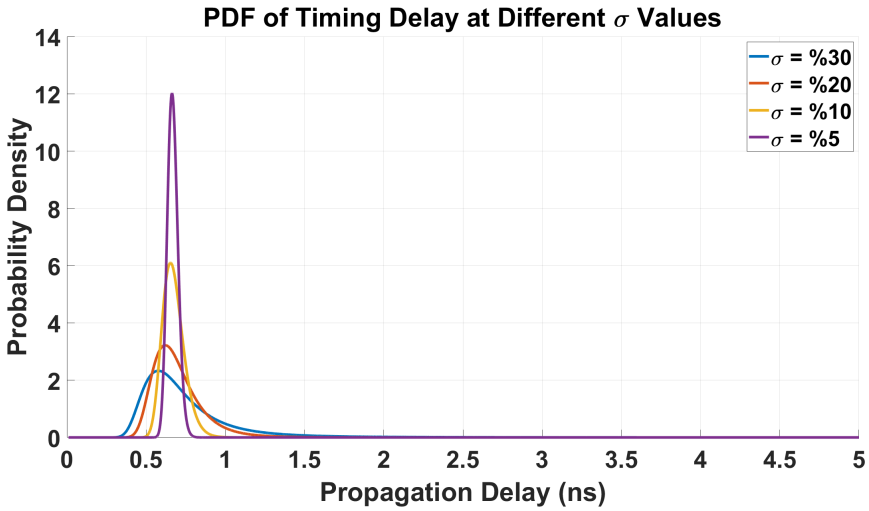

Figure 2.10 Timing PDF

In a synchronous digital circuit, a data path fails if the total delay through the combinational logic (t_G) plus the setup time (t_S) exceeds the clock period (T_{clk}).

$$t_G + t_s > T_{clk}$$

Since timing varies due to voltage variation, error probability of timing failure in

given the path, can be calculated based on the timing PDF. Assuming that example circuit works at 1Ghz clock period, to be able to calculate the error probability, area under the timing PDF as depicted on Figure 2.11 should be calculated. (1Ghz clock speed is chosen since setup circuit consist of 5 logic gates and 2 D-Flip-Flops (DFF) which is small.)

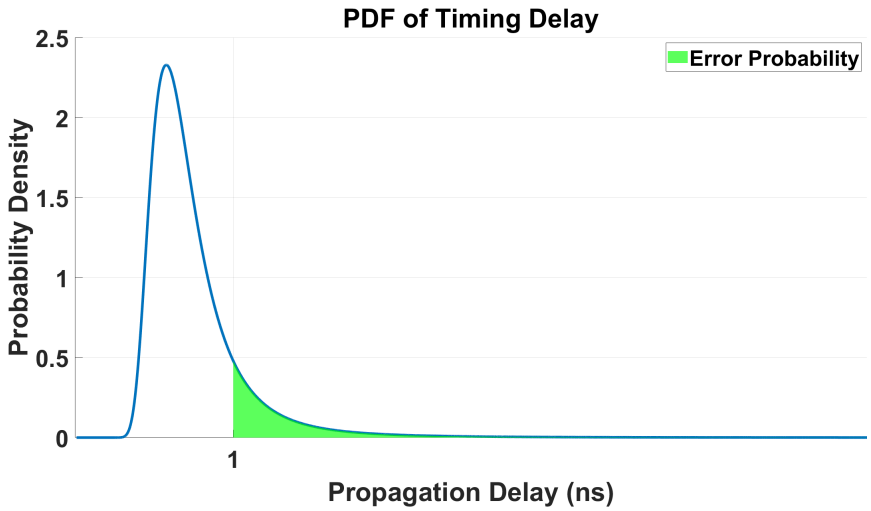

Figure 2.11 Timing PDF

This area can be expressed using the integration 2.4 and expression represents the total area under the timing PDF where the data path delay surpasses the clock period, thereby indicating a timing error.

$$EP = \int_{T_{clk}}^{\infty} p(\delta)\, d\delta \tag{2.4}$$

$p(\delta)$ represents the timing PDF, and the lower integration limit T_{clk} is the clock period (Rathore et al. (2020)).

Since voltage PDF corresponds timing PDF, so voltage PDF can also be used for the error probability calculation. The lowest voltage level that meets timing requirement without any violation is the operational limit for the circuit. For the given circuit, minimum $636mV$ satisfies the timing requirement which is $t_G + t_s < 1ns$.

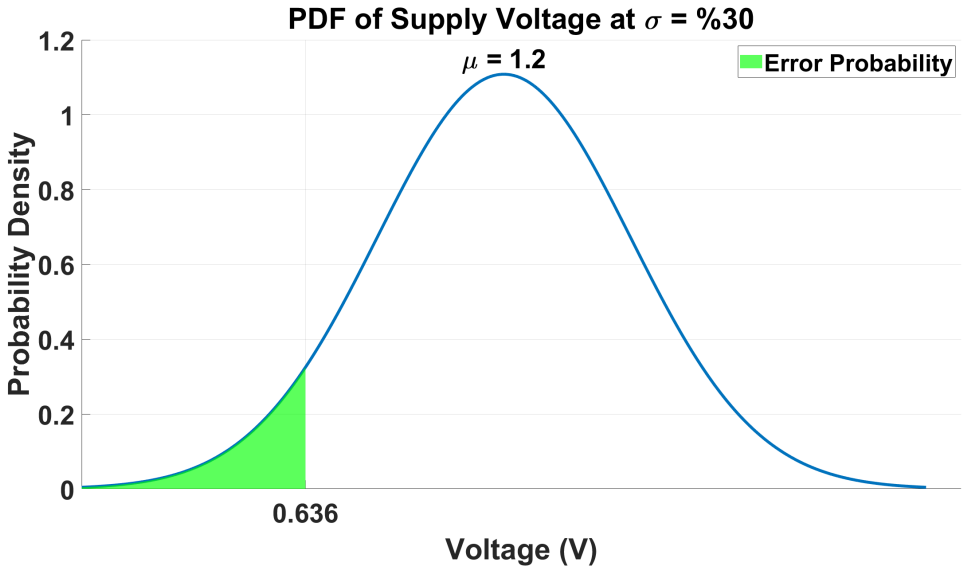

Figure 2.12 Voltage PDF

In Equation 2.5, voltage PDF utulized to calculate error probability. $p(V)$ represents the voltage PDF, and the upper integration limit V_{max} corresponds to minimum voltage level that circuit does not have any timing violation.

$$EP = \int_{-\infty}^{V_{max}} p(V)\, dV \tag{2.5}$$

Both shaded area corresponds to the same error probability and since delay monotonically decreases when voltage increases. The integration 2.5 is used for the calculation of error probability for the given sequential circuit for further analysis. To verify the model, Monte Carlo (MC) simulation is performed with 5000 samples and only VDD is varied with a Gaussian distribution with different σ and μ values.

2.4.1.1 Discussion

The error probabilities (%) are depicted in Table 2.1 for different voltage conditions. There is small difference between MC simulation which only VDD varies, and proposed error probability calculation method. Magnitude of error probability difference between MC simulations and the method, changes with respect to given voltage characteristic. That is directly related with randomness of the MC simulation.

Method / Voltage Characteristic	MC Simulation (V_{DD} Vary)	V Model STA (V_{DD} Sweep)
$V_{DD} = 1.2V$ $\sigma = \%10$	0	$7.6097x10^{-5}$
$V_{DD} = 1.2V$ $\sigma = \%30$	5.54	5.4492
$V_{DD} = 0.9V$ $\sigma = \%10$	0.08	0.1043
$V_{DD} = 0.9V$ $\sigma = \%30$	15.44	15.2463
$V_{DD} = 0.6V$ $\sigma = \%10$	65.3867	64.9264
$V_{DD} = 0.6V$ $\sigma = \%30$	55.5467	55.0838

Table 2.1 Probabilistic Model vs MC (Voltage Variation) ($1Ghz$)

Probabilistic approach can be used for the MC simulations to find error probability with a small difference. Since probabilistic timing model avoids transistor level simulation which requires high amount of time.

2.4.2 Error Probability for Multiple Timing Paths

The timing error probability for a sequential data path with a single input is examined in section 2.4.1. In this section, the methodology is expanded to calculate the timing error probability for multiple timing paths within the same pipeline stage. It is assumed that the average supply voltage of the gates in the same pipeline stage is uniform. This assumption is based on the observation that adjacent nodes exhibit a high spatial correlation in average supply voltage (Rathore et al. (2020)). Hence, to determine the probability of timing error at the output, driven by various data paths, integrating the voltage PDF is applied as previously discussed. The upper limit of integration is set by the highest voltage which at least one of the timing path experiences a failure (Rathore et al. (2020)). Critical path has the highest timing delay. Therefore, the error probability of the output is determined by the error probability of the critical path.

2.4.3 Error Probability for Sequentially Adjacent Paths

Figure 2.13, 1bit MAC unit is illustrated (Rathore et al. (2020)) for multiple paths within a pipeline stage and sequentially adjacent pipeline stages. The probability of a timing error in *out*1 driven by multiple paths is calculated by integrating the voltage PDF as previously explained.

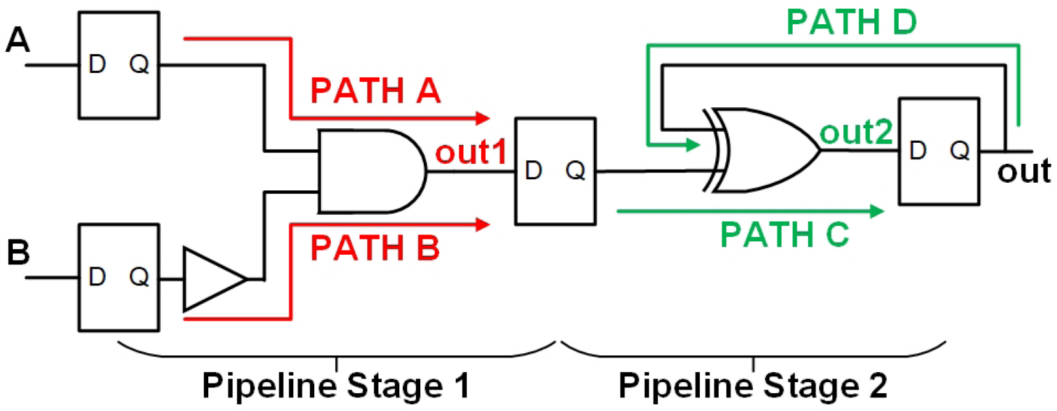

Figure 2.13 1-Bit MAC Unit (Rathore et al., 2020)

The corresponding voltage and timing PDF of $PATH\,A$ and $PATH\,B$ are shown in Figure 2.14.

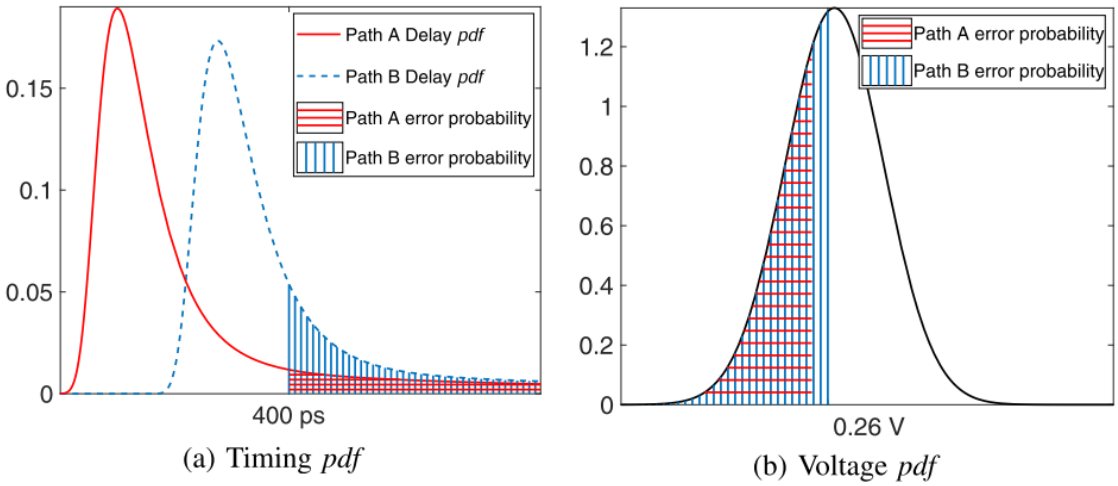

Figure 2.14 PDF of Multiple Paths (Rathore et al., 2020)

Only AND gate contributes the timing delay of $PATH\,A$ whereas BUFFER and AND gates contribute the timing delay of $PATH\,B$. Therefore, the upper limit

voltage is determined by the $PATH\,B$ since it has more path delay for that pipeline stage.

Assuming the supply voltage for different pipeline stages is independent, the timing error probability for each stage can be calculated separately. Error at the output can be the result of failure in any stage or simultaneous failures in multiple stages. So, the overall error probability at the final output node can be determined by combining the error probability of each pipeline stage.

$$\begin{aligned} P_{out} &= P_{out1} \cup P_{out2} \\ P_{out} &= P_{out1} + P_{out2} - P_{out1} * P_{out2} \end{aligned} \tag{2.6}$$

To calculate the error probability at the *out*, union operation is utilized as depicted in Equation 2.6. Union operation is used for the calculation of total error probability for the output of the MAC array that explained at the section 2.4.7.

2.4.4 Timing Probability Distribution Considering Temperature

Silicon chips, composed of semiconductor material and metal layers that interconnect transistors, experience changes in performance with temperature fluctuations. As temperature rises, the carrier mobility within the semiconductor decreases, leading to a reduction in conductivity (Pierret (1996)). Conversely, the threshold voltage V_{TH} of the transistors also decreases with rising temperatures (Pierret (1996)), which can have complex effects on total performance.

Additionally, the conductivity of the metal layers diminishes as temperature increases (Kittel (2007)). These factors together mean that the overall impact of temperature on chip performance depends on the operating voltage. Temperature changes can either enhance or degrade the chip's performance depending on the specific conditions.

Operating temperature of the chip changes with the environment temperature, clock frequency, and workload of the chip. The variation in temperature can be modeled as a random variable in a Gaussian distribution similar to the voltage variation. The standard deviation (σ) represents the temperature variation due to workload of the operations and other effects, while the mean (μ) corresponds to temperature when chip works under moderate workload as depicted in Figure 2.15.

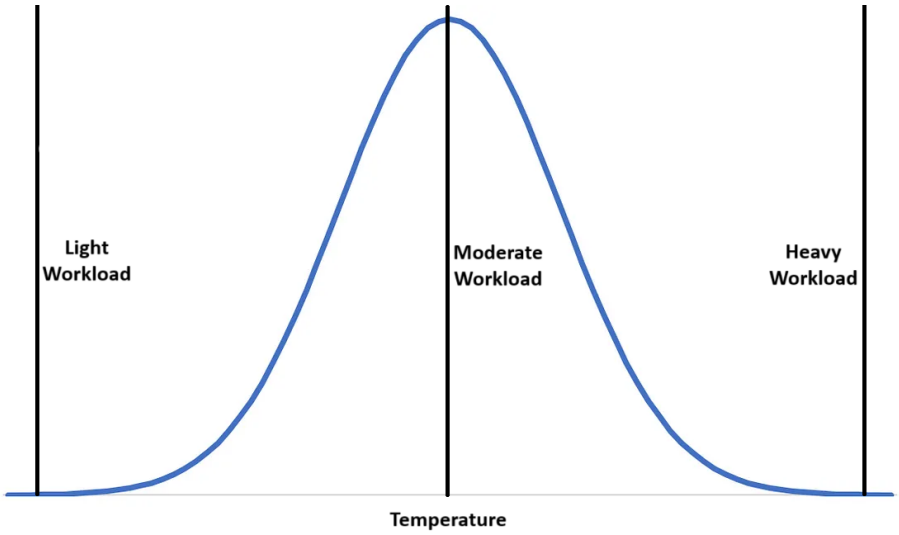

Figure 2.15 Temperature Variation Dependency of the Chip States

Temperature-Voltage Delay Characteristic of the circuit 2.7 is given in Figure 2.16. At lower voltages, circuit gets faster when temperature increases, however reverse effect is observed at higher voltages. In the middle such as $0.9V$, until certain temperature, circuit gets faster when temperature increases, further increase in temperature makes circuit slower. Regardless of the temperature, rise in voltage improves circuit performance meaning reduces the delay.

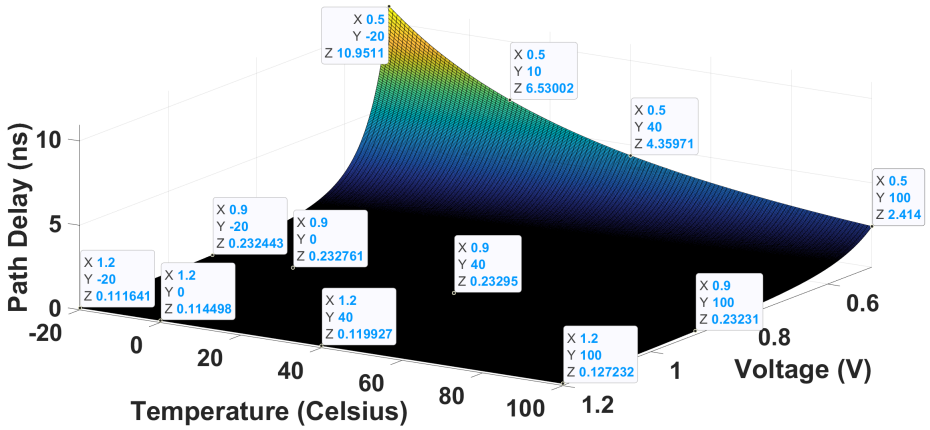

Figure 2.16 Voltage & Temperature versus Delay
(X:Voltage, Y:Temperature, Z:Delay)

Some of the voltage-temperature-delay points are shown in Figure 2.16. Temperature-delay relations are shown for supply voltages $0.6V$, $0.9V$ and $1.2V$ in Figures 2.17, 2.18, 2.19 respectively. There is no direct relation between delay and temperature as opposed to voltage-delay relation.

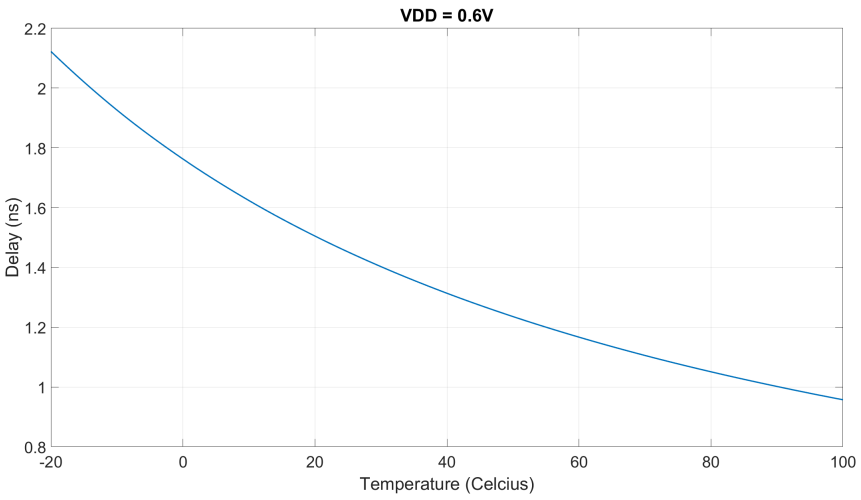

Figure 2.17 Temperature-Delay Characteristic at $0.6V$

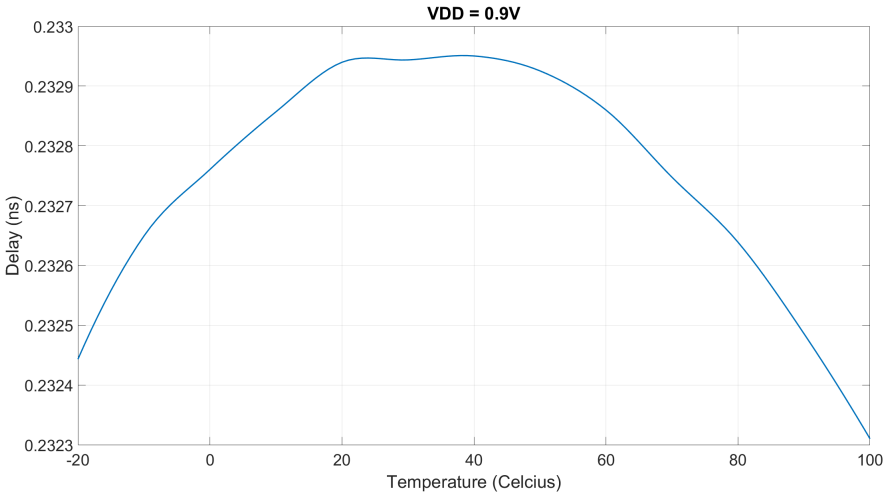

Figure 2.18 Temperature-Delay Characteristic at $0.9V$

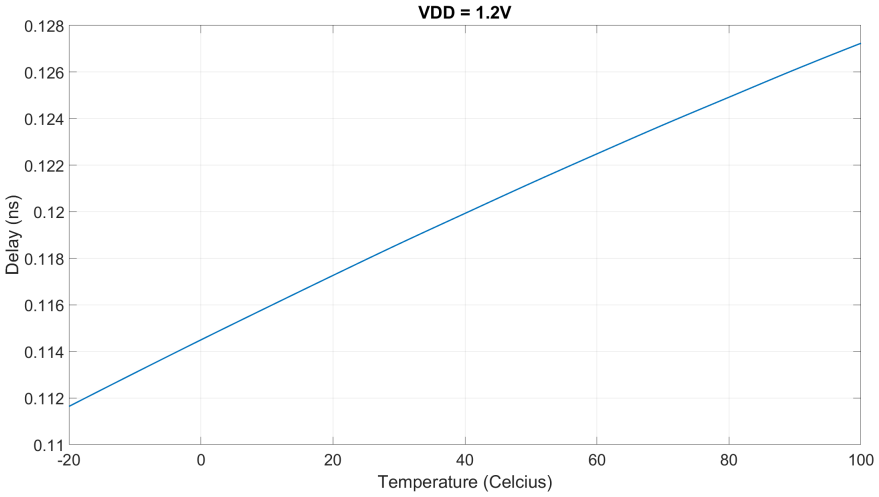

Figure 2.19 Temperature-Delay Characteristic at $1.2V$

So, temperature effect is considered locally and probabilistic error calculation algorithm modified as given in Equation 2.7,

$$EP = \sum_{T_n=-273}^{\infty} \Big(\int_{T_n}^{T_n+T_{res}} p(T)\, dT \Big), \text{ where } p(T_n) > \text{ Clock period.} \tag{2.7}$$

Practically, temperature range between $-20°C$ and $100°C$ are used for analysis. T_{res} corresponds to resolution and $1°C$ is used for the error probability calculation. Temperature aware model compares the timing delay with the clock period at the current temperature. If failure occurs, than that temperature contributes the error probability. If the resolution temperature decreases, model becomes more accurate with the trade of increase in analysis time.

2.4.4.1 Discussion

MC simulation and temperature aware model are compared and error probabilities (%) are given in Table 2.2 for different temperature variations. Nominal temperature $40°C$ is selected based on the study that explores the thermal characterization of the heterogeneous MPSoC (Iranfar et al. (2017)) and variation is approximately $10°C$. Another study shows that temperature of chip can go up to $90°C$ depending on the workload (Zhang, Sadiqbatcha & Tan (2023)). For the comparison, high temperature variation is performed with the $\sigma = 30°C$. Additionally, comparison is done at different voltages since temperature effect behaves differently depending on the voltage. Magnitude of the temperature variation depends on different factors such as operating voltage, clock frequency, cooling down methodology of chip etc. Therefore, magnitude of the temperature variations are selected empirically. When voltage level higher than the $0.7V$, error probability is 0 for given temperature characteristic, so that they are not shown in the table. $0.7V$ and lower voltages are used to compare the MC simulation and temperature aware model.

Temperature plays a critical role especially when there is a little margin for the timing specification. For example, at $0.65V$ depending on the noise level of temperature, error probability increases significantly since it is the voltage level that timing specification has little margin. MC Simulation and T Model STA has significant differences when noise of the temperature is high. That means, accuracy of characterization of the gates with respect to temperature, becomes even more

Temperature Characteristic \ Method	MC Simulation (Temperature Vary)	T Model STA
$V_{DD} = 0.7V$ $T = 40°C$, $\sigma = 10°C$	0	0
$V_{DD} = 0.7V$ $T = 40°C$, $\sigma = 30°C$	0	0
$V_{DD} = 0.65V$ $T = 40°C$, $\sigma = 10°C$	0.04	0.0337
$V_{DD} = 0.65V$ $T = 40°C$, $\sigma = 30°C$	12.4433	5.12
$V_{DD} = 0.6V$ $T = 40°C$, $\sigma = 10°C$	100	99.8134
$V_{DD} = 0.6V$ $T = 40°C$, $\sigma = 30°C$	95.4567	83.3145

Table 2.2 Probabilistic Model vs MC Simulation (Temperature Variation) ($1Ghz$)

significant at low supply voltages.

This method provides fast observation with high magnitude of error with respect to actual MC simulation. The model's temperature awareness must be improved by improving gate characterization method.

2.4.5 Voltage and Temperature Varied Timing Distribution

Considering both voltage and temperature effect together requires multivariate analysis for the timing distribution. Apart from the voltage and temperature μ and σ values, their correlation (ρ) is necessary to calculate error probability of the path. Power dissipation causes heat dissipation to environment that result in temperature increase on the chip. Both voltage and operating frequency contributes power consumption as shown in Equation 2.8 (Kang & Leblebici (2003)).

$$P_{avg} = C_{load} * {V_{DD}}^2 * f \tag{2.8}$$

Therefore, depending on the cooling down methodology of the chip and environment, voltage level and frequency have positive correlation with the temperature of the chip. However finding magnitude of the correlation constant (ρ), requires advanced heat transfer analysis tool or measurement. For simplicity, correlation of temperature between both voltage and frequency is assumed to be 0 which means, temperature and voltage are independent variables for further analysis.

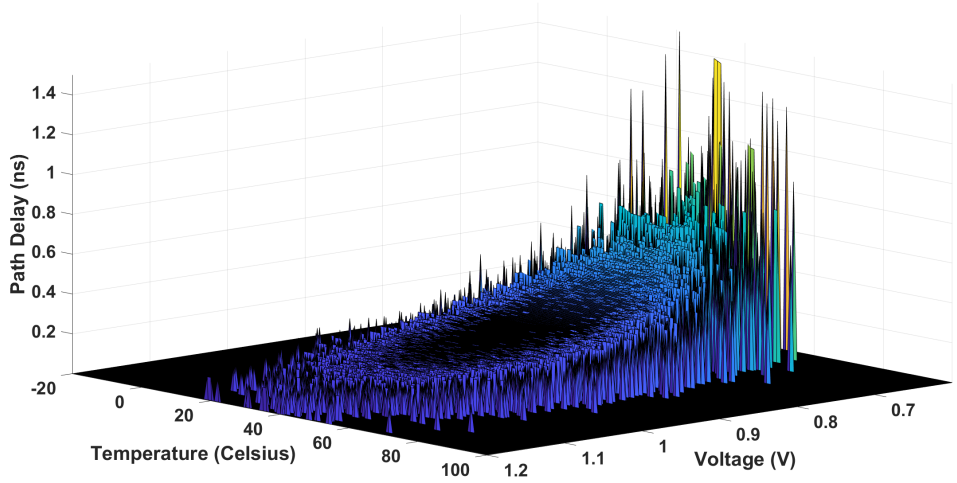

Figure 2.20 MC Simulation Timing Distribution where,
$\mu_V = 0.9V$, $\mu_T = 40°C$, $\sigma_V = 0.09V$, $\sigma_T = 10°C$, $\rho = 0$

Figure 2.20 shows the delay distribution of MC simulation result with given voltage and temperature conditions. Z axes shows the delay of corresponding voltage and temperature (not the frequency of the corresponding voltage, temperature pair). Figure 2.21 shows the PDF of voltage and temperature combination with assuming $\rho = 0$.

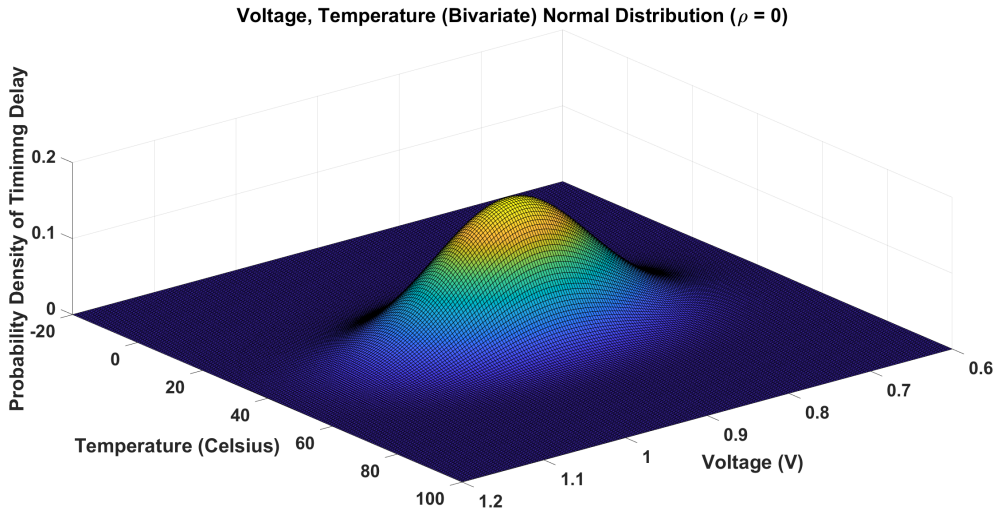

Figure 2.21 PDF of Timing Delay where,
$\mu_V = 0.9V$, $\mu_T = 40°C$, $\sigma_V = 0.09V$, $\sigma_T = 10°C$, $\rho = 0$

The volume under the points where *path delay* > *period* is gives the error probability. To calculate the error probability, both voltage and temperature error probability

calculations are combined and the following formula is used to calculate error probability with given,

μ_T, σ_T

μ_V, σ_V

ρ between temperature and voltage

(2.9)

$$EP = \sum_{T_n=-273}^{\infty} \left(\int_{T_n}^{T_n+T_{res}} \left(\int_{-\infty}^{V_{max}} p(V,T)\, dV \right) dT \right), \text{ where } p(V_{max}, T_n) > \text{ Clock period.}$$

V_{max} represents the maximum voltage value that causes timing error on the path at temperature T_n. Regardless of the temperature value, voltage increase reduces the propagation delay as shown previously in Figure 2.16. Therefore, for each temperature, V_{max} value is found and at that temperature, and then the volume under the voltage-temperature distribution is found at interval $(-\infty, V_{max}]$. Then, all of the volumes at different temperatures are added and that gives the error probability.

2.4.5.1 Discussion

MC simulation and VT aware model are compared and error probabilities (%) are given in Table 2.3 for different voltage and temperature characteristics.

Method / VT Characteristic	MC Simulation (V_{DD} *and* T Vary)	VT Model STA
$V_{DD} = 1.2V$, $\sigma = \%10$ $T = 40°C$, $\sigma = 10°C$	0	$6.2086x10^{-5}$
$V_{DD} = 1.2V$, $\sigma = \%30$ $T = 40°C$, $\sigma = 30°C$	5.4545	4.3744
$V_{DD} = 0.9V$, $\sigma = \%10$ $T = 40°C$, $\sigma = 10°C$	0.1033	0.0865
$V_{DD} = 0.9V$, $\sigma = \%30$ $T = 40°C$, $\sigma = 30°C$	15.2133	12.2344
$V_{DD} = 0.6V$, $\sigma = \%10$ $T = 40°C$, $\sigma = 10°C$	69.0033	61.1809
$V_{DD} = 0.6V$, $\sigma = \%30$ $T = 40°C$, $\sigma = 30°C$	54.38	44.1466

Table 2.3 Probabilistic Model vs MC Simulation (Voltage and Temperature Variation) ($1Ghz$)

From the results, VT Model STA error probability is the lower then the MC simulation error probability for all given characteristics. As a result, accuracy of the gate's timing delay extraction gains more significance because MC Simulations also require

transistor level simulation. Transistor level simulation and STA method might differ in terms of timing delay of the given path even in the same voltage-temperature condition. However, VT Model STA provides error probability of the path faster than the among all methods. Therefore, error probability of path can be found by using the formula that is given in Equation 2.9 instead of long MC simulations. This result can be used to observe bit error probability on the circuit intuitively since it provides optimistic timing error probability for all voltage-temperature characteristic.

2.4.6 Process Variation Integration

Process parameters are randomly chosen by the simulator while performing MC simulation. They can't be swept as Voltage or Temperature value. Propagation delay variation due to process variation, was extracted by performing MC simulations on process parameters. Since magnitude of the variation differs at different voltages and temperatures, MC simulations were performed at different voltage-temperature combinations. In order to reduce the number of simulations, voltage and temperature points were chosen with resolution of $150mV$ and $30°C$ respectively. Propagation delays obtained from MC simulations were used to find the standard deviation (σ) of the propagation delays. Thus, at that corner, standard deviation of propagation delay can be used to considering process variation without performing MC simulations.

$1.2V$, $-20°C$	$1.2V$, $10°C$	$1.2V$, $40°C$	$1.2V$, $70°C$	$1.2V$, $100°C$
$1.05V$, $-20°C$	$1.05V$, $10°C$	$1.05V$, $40°C$	$1.05V$, $70°C$	$1.05V$, $100°C$
$0.9V$, $-20°C$	$0.9V$, $10°C$	$0.9V$, $40°C$	$0.9V$, $70°C$	$0.9V$, $100°C$
$0.75V$, $-20°C$	$0.75V$, $10°C$	$0.75V$, $40°C$	$0.75V$, $70°C$	$0.75V$, $100°C$
$0.6V$, $-20°C$	$0.6V$, $10°C$	$0.6V$, $40°C$	$0.6V$, $70°C$	$0.6V$, $100°C$

Table 2.4 Corners that Process Variation Extracted

Table 2.4 shows the voltage, temperature pairs that process variation extraction was performed. Then, the table was used for 2D interpolation to increase the resolution of the voltage and temperature. Lastly, standard deviation lookup table is used to create new propagation delay lookup tables as shown in Figure 2.22.

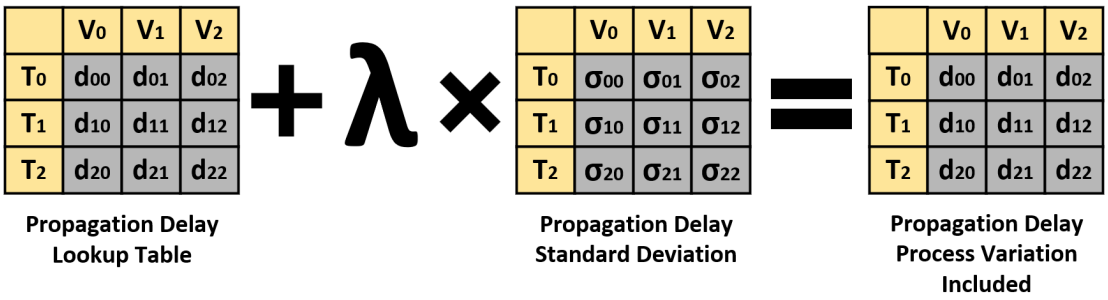

Figure 2.22 Propagation Delay Inclusion Process Variation

λ is selected based on the Gaussian distribution with $\mu = 0$ and $\sigma = \sigma_0$ where sigma σ_0 corresponding percentage. For instance, %10 corresponds to $\sigma_0 = 0.1$ (At higher voltages voltages, σ_0 is approximately %10 whereas at lower voltages, σ_0 reaches up to %30). For pessimistic approach, σ_0 should be selected higher. To exacerbate propagation delay further, taking absolute value of the λ results that propagation delay always increases the due to process variations. In the silicon chip, adjacent gates are affected with similar process variation, so that the same λ value is used for creating the new lookup table. This requires the assumption that, each gate is affected by the process variation in a similar way in terms of propagation delay which might not be the case in real life.

2.4.6.1 Discussion

MC simulation and Process Model STA are compared in terms of error probabilities (%) at the corners. At each corner, MC simulations with 1500 sample are performed. To be able compare it with the methodology, 1500 new timing path delay created with explained method. When voltage value higher than $0.9V$, error probabilities are 0 that are not shown on the Figure 2.23.

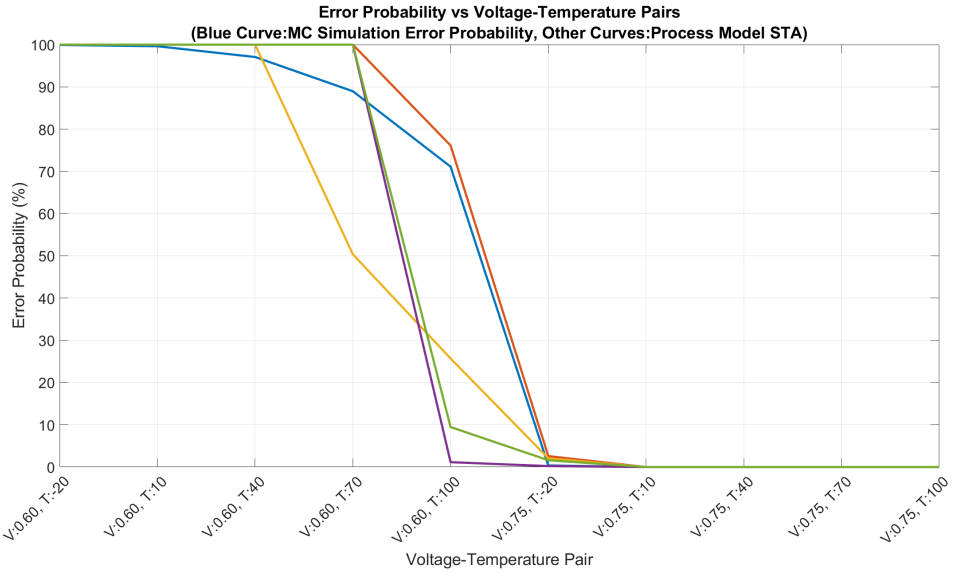

Figure 2.23 Error Probability at the Corners ($1Ghz$)

As depicted in the figure, blue curve corresponds process varied MC simulations. The other colors are repeated result of Process Model STA analysis to show error probabilities are changing due to randomness of the process variation integration into model. Both MC simulation and model STA results follow a similar trend except their magnitudes are different. At the $0.6V$ both have high error probability. In Table 2.5, error probabilities (%) are shown for the some corners.

Corner	MC Simulation Error Probability	Model STA 1	Model STA 2	Model STA 3	Model STA 4
$0.6V$, $-20°C$	99.925	100	100	100	100
$0.6V$, $10°C$	99.625	100	100	100	100
$0.6V$, $40°C$	97.074	100	100	100	100
$0.6V$, $70°C$	88.972	100	50.369	100	100
$0.6V$, $100°C$	71.117	76.125	25.701	1.154	9.468
$0.75V$, $-20°C$	0.375	2.592	2.067	0.231	1.616
$0.75V$, $10°C$	0.075	0.955	0	0	0

Table 2.5 Error Probability Comparison at Corners (EP Different than 0)

At some corners, MC error probability is higher than the model error probability and reverse is true for other corners. This difference occurs due to the randomness of the MC simulation and also model calculation as explained previously.

2.4.7 PVT Aware Probabilistic Timing Error Probability Calculation

Error probability is calculated based on the 3 variables as previously explained to calculate error probability by taking into account all effects. Initially, path delay is calculated with the method that explained at section 2.4.6. Then error probability is calculated as explained section 2.4.5.

Method / VT Characteristic	MC Simulation (PVT Vary)	PVT Model STA
$V_{DD} = 1.2V$, $\sigma = \%10$ $T = 40°C$, $\sigma = 10°C$	0	$6.2493x10-5$
$V_{DD} = 1.2V$, $\sigma = \%30$ $T = 40°C$, $\sigma = 10°C$	5.6049	4.3919
$V_{DD} = 0.9V$, $\sigma = \%10$ $T = 40°C$, $\sigma = 10°C$	0.2144	0.0874
$V_{DD} = 0.9V$, $\sigma = \%30$ $T = 40°C$, $\sigma = 10°C$	12.5431	12.292
$V_{DD} = 0.6V$, $\sigma = \%10$ $T = 40°C$, $\sigma = 10°C$	68.0461	61.201
$V_{DD} = 0.6V$, $\sigma = \%30$ $T = 40°C$, $\sigma = 10°C$	52.1936	44.1862

Table 2.6 Probabilistic Model vs MC Simulation (PVT Variation) ($1Ghz$)

For all conditions, PVT Model STA calculates lower error probability with respect to MC simulations as shown on Table 2.6. However as previously discussed, due to nature of the randomness, error probability varies.

An actual case study was conducted on a Multiply-Accumulate (MAC) unit adder circuit. In the MAC unit, Ripple Carry adder architecture was implemented. One of the reason for choosing the Ripple Carry adder architecture was, its ability to easily observe timing errors with given input combinations. To compare PVT Model STA with MC simulations, the signal propagation of each bit must be observed, and this requires generating propagation through specific input combinations. Since multiplier has false paths and choosing correct input combination to trigger the longest path is difficult, multiplier was not included in the analysis.

2.4.7.1 Results

Probabilistic model calculates the error probability independent from the input combinations, however for MC simulation, an input combination must be provided to circuit and that input combination must trigger critical path to observe error probability of all bits. In order to satisfy this condition, one input is selected as 1 and the

other one selected as -1. In 2's complement addition, signal transition throughout the critical path has occurred.

Error probabilities are compared at 200 Mhz since MAC units are designed to work at 200 Mhz clock frequency. 3 different voltage value which are $1.1V$, $1V$ and $0.9V$ are compared in terms of error probability. Analysis at lower voltages are not shown due to high error probability.

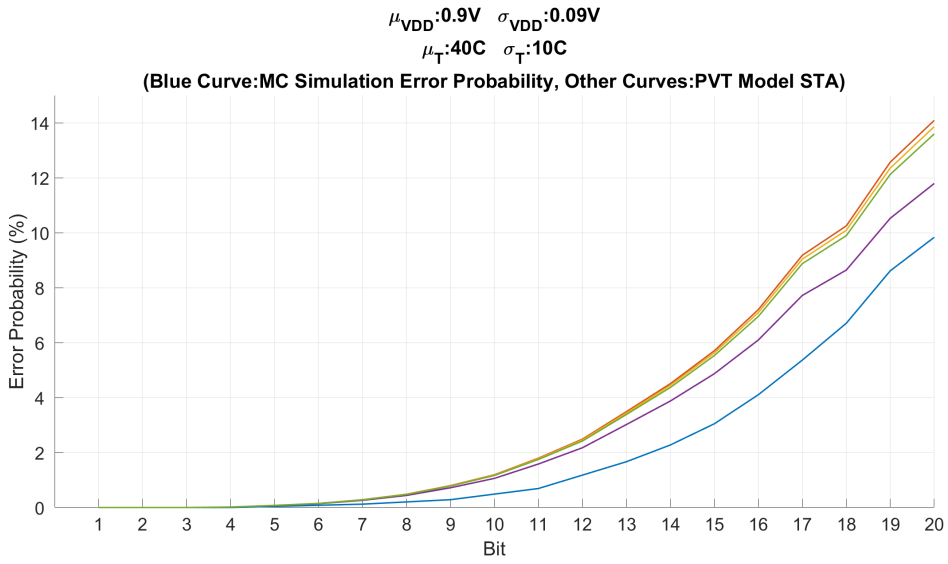

Figure 2.24 Error Probability 20 Bit Ripple Carry Adder ($200Mhz$)

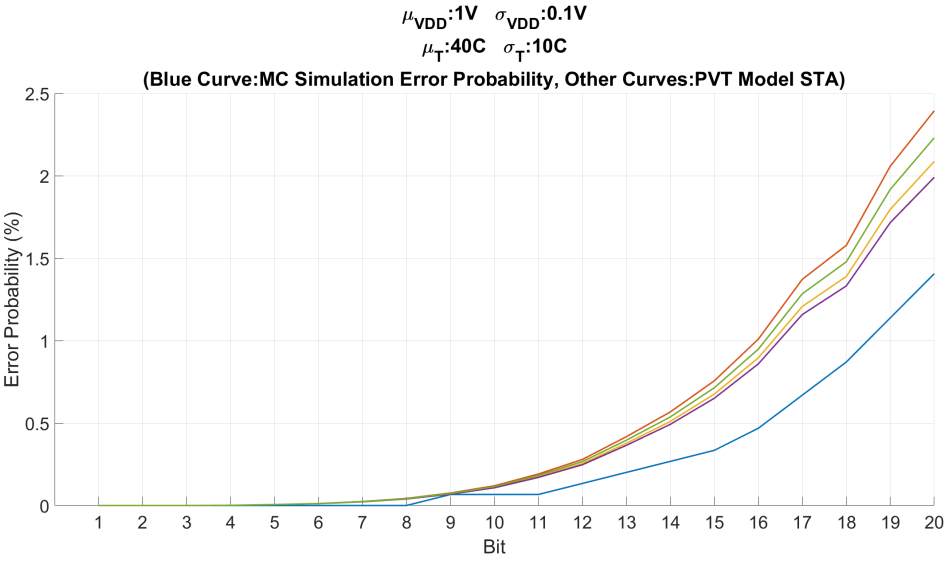

Figure 2.25 Error Probability 20 Bit Ripple Carry Adder ($200Mhz$)

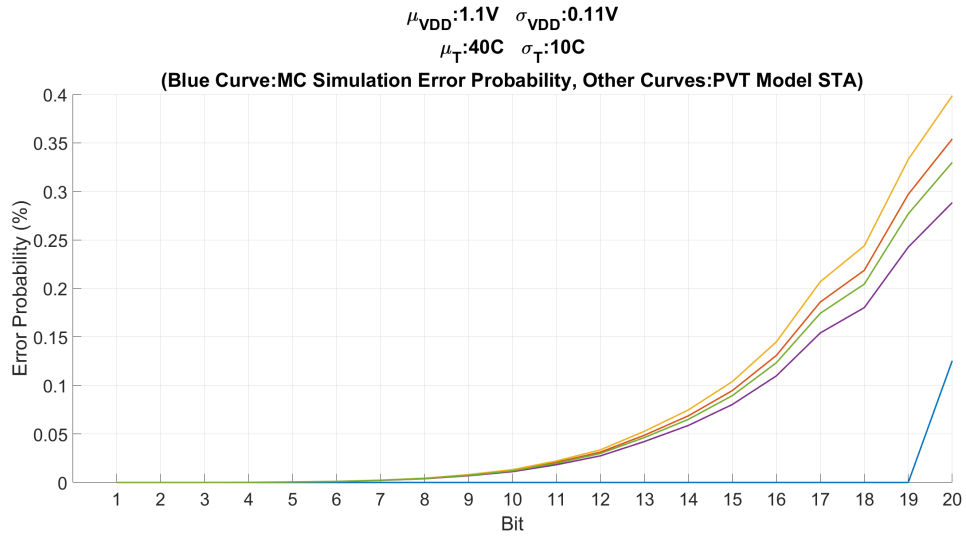

Figure 2.26 Error Probability 20 Bit Ripple Carry Adder ($200Mhz$)

In Figures 2.24, 2.25 and 2.26, the blue curve shows the error probability of MC simulation, other curves represent error probability of the PVT aware model analysis. As shown in all figures, the model calculates the error probability higher than the MC simulations. In the previous analysis of small circuit, the error probabilities are lower then the MC simulation analysis for almost all of the given VT characteristics. From this observation, when the given netlist gets larger, PVT aware model becomes more pessimistic. This is the result of transient simulation and STA difference. That makes the analysis conservative for large netlists and makes less reliable for the small netlists. However the gap between model and MC simulation could be reduced by improving the analysis (discussed at future work) and that brings additional overhead for the model.

Also another observation is, result of the MC simulation in Figure 2.25, error probability of Bit 9 and 10 are higher then the error probability of Bit 11 which shouldn't be the case for the Ripple Carry Adder architecture since higher bits have higher path delay. However, with the MC simulation, gates are affected differently. Some gates get faster whereas others might become slower. In the model analysis, all of the gates in the same pipeline are assumed to be affected the same manner in terms of propagation delay. That also reduces the accuracy when netlist is large.

Model STA	MC Simulation (1500 sample)
49 second	11 hour

Table 2.7 Probabilistic Model vs MC Simulation Spent Time

For time comparison, model analysis for all bits takes approximately 49 second whereas MC simulation takes about 27 second for one sample simulation. Depending

on the number of samples, overall MC simulation might take several hours. For confidence, MC simulations are performed with 1500 samples and took about 11 hours. Thus, probabilistic model analysis makes the test roughly 808 times faster than the MC simulation with a decent error probability accuracy. For the large netlist, model error probability is more pessimistic than the MC simulation, voltage supply level can be regulated based on the analysis.

In the Accelerator, MAC Units are identical, and each adder circuit has timing error probability. Since multiplier has lots of false paths, analysis of error probability contribution by the multiplier discarded and error probability assumed to be 0. Based on the analysis of multiple timing path error probability (2.4.2), total timing error probability of result is calculated by the following code,

Listing 2.1 MAC Unit Error Probability

```
EPmac = EPmult + EPadder - EPmult * EPadder;

EPtotal = 0;
for (int k = 0; k < 16; k++){
    EPtotal = EPtotal + EPmac - EPtotal * EPmac;
}
```

EP_{mac} and EP_{total} of result is calculated and MSB of the result error probability are depicted in Tables 2.8 and 2.9 for different voltage, temperature characteristics.

As it can be seen from the tables, accumulation error probability blows up. This is the result of the accumulation of the error probability, as depicted on method 2.1. However, this is the result based on the critical path signal transition which may not be always the case with given input combinations. Actual error probability might be less than the probabilistic error model analysis.

Supply Voltage and Temperature Characteristic	MAC Unit Error Probability (%)	Accumulated Error Probability of Array (%)
$V_{DD} = 0.9V \ T = 40°C$ $\sigma_{V_{DD}} = \%5 \ \sigma_T = 10°C$	0.7829	14.5473
$V_{DD} = 0.9V \ T = 40°C$ $\sigma_{V_{DD}} = \%10 \ \sigma_T = 10°C$	11.8152	91.9114
$V_{DD} = 0.9V \ T = 40°C$ $\sigma_{V_{DD}} = \%5 \ \sigma_T = 30°C$	0.7716	14.3516
$V_{DD} = 0.9V \ T = 40°C$ $\sigma_{V_{DD}} = \%10 \ \sigma_T = 30°C$	11.8373	91.9519
$V_{DD} = 0.9V \ T = 10°C$ $\sigma_{V_{DD}} = \%5 \ \sigma_T = 10°C$	1.2818	22.7422
$V_{DD} = 0.9V \ T = 10°C$ $\sigma_{V_{DD}} = \%10 \ \sigma_T = 10°C$	13.4405	94.4243
$V_{DD} = 0.9V \ T = 10°C$ $\sigma_{V_{DD}} = \%5 \ \sigma_T = 30°C$	1.8991	31.8511
$V_{DD} = 0.9V \ T = 10°C$ $\sigma_{V_{DD}} = \%10 \ \sigma_T = 30°C$	12.6792	93.3573
$V_{DD} = 0.95V \ T = 40°C$ $\sigma_{V_{DD}} = \%5 \ \sigma_T = 10°C$	0.04066	0.8101
$V_{DD} = 0.95V \ T = 40°C$ $\sigma_{V_{DD}} = \%10 \ \sigma_T = 10°C$	5.0650	64.6392
$V_{DD} = 0.95V \ T = 40°C$ $\sigma_{V_{DD}} = \%5 \ \sigma_T = 30°C$	0.0538	1.0709
$V_{DD} = 0.95V \ T = 40°C$ $\sigma_{V_{DD}} = \%10 \ \sigma_T = 30°C$	5.0726	64.6957
$V_{DD} = 0.95V \ T = 10°C$ $\sigma_{V_{DD}} = \%5 \ \sigma_T = 10°C$	0.0956	1.8954
$V_{DD} = 0.95V \ T = 10°C$ $\sigma_{V_{DD}} = \%10 \ \sigma_T = 10°C$	5.1919	65.5723
$V_{DD} = 0.95V \ T = 10°C$ $\sigma_{V_{DD}} = \%5 \ \sigma_T = 30°C$	0.1265	2.5014
$V_{DD} = 0.95V \ T = 10°C$ $\sigma_{V_{DD}} = \%10 \ \sigma_T = 30°C$	5.2030	65.6532

Table 2.8 Calculated Error Probability ($200Mhz$)

As it can be seen from the Tables 2.8 and 2.9, voltage supply level is dominant, and its noise level affects the error probability greater than the temperature noise level.

Supply Voltage and Temperature Characteristic	MAC Unit Error Probability (%)	Accumulated Error Probability of Array (%)
$V_{DD} = 1V\ T = 40°C$ $\sigma_{V_{DD}} = \%5\ \sigma_T = 10°C$	0.0018	0.0372
$V_{DD} = 1V\ T = 40°C$ $\sigma_{V_{DD}} = \%10\ \sigma_T = 10°C$	1.8823	31.6182
$V_{DD} = 1V\ T = 40°C$ $\sigma_{V_{DD}} = \%5\ \sigma_T = 30°C$	0.0018	0.0372
$V_{DD} = 1V\ T = 40°C$ $\sigma_{V_{DD}} = \%10\ \sigma_T = 30°C$	1.8795	31.5783
$V_{DD} = 1V\ T = 10°C$ $\sigma_{V_{DD}} = \%5\ \sigma_T = 10°C$	0.0038	0.0768
$V_{DD} = 1V\ T = 10°C$ $\sigma_{V_{DD}} = \%10\ \sigma_T = 10°C$	1.8438	31.0790
$V_{DD} = 1V\ T = 10°C$ $\sigma_{V_{DD}} = \%5\ \sigma_T = 30°C$	0.0051	0.1026
$V_{DD} = 1V\ T = 10°C$ $\sigma_{V_{DD}} = \%10\ \sigma_T = 30°C$	2.5627	40.5024
$V_{DD} = 1.05V\ T = 40°C$ $\sigma_{V_{DD}} = \%5\ \sigma_T = 10°C$	$5.5419x10^{-5}$	0.0011
$V_{DD} = 1.05V\ T = 40°C$ $\sigma_{V_{DD}} = \%10\ \sigma_T = 10°C$	0.7143	13.3584
$V_{DD} = 1.05V\ T = 40°C$ $\sigma_{V_{DD}} = \%5\ \sigma_T = 30°C$	$8.4806x10^{-5}$	0.0017
$V_{DD} = 1.05V\ T = 40°C$ $\sigma_{V_{DD}} = \%10\ \sigma_T = 30°C$	0.6801	12.7586
$V_{DD} = 1.05V\ T = 10°C$ $\sigma_{V_{DD}} = \%5\ \sigma_T = 10°C$	$9.5801x10^{-5}$	0.0019
$V_{DD} = 1.05V\ T = 10°C$ $\sigma_{V_{DD}} = \%10\ \sigma_T = 10°C$	0.8234	15.2418
$V_{DD} = 1.05V\ T = 10°C$ $\sigma_{V_{DD}} = \%5\ \sigma_T = 30°C$	0.0001	0.0024
$V_{DD} = 1.05V\ T = 10°C$ $\sigma_{V_{DD}} = \%10\ \sigma_T = 30°C$	0.8929	16.4217

Table 2.9 Calculated Error Probability ($200Mhz$)

With the consideration of the error probability, supply voltage can be reduced depending on the supply noises, application, DNN model, or layer to save power with the increasing error probability trade-off. Power consumption and error probability graph with respect to voltage value is shown in Figure 2.27 (dynamic power consumption) and Figure 2.28 (static power consumption) at different noise value.

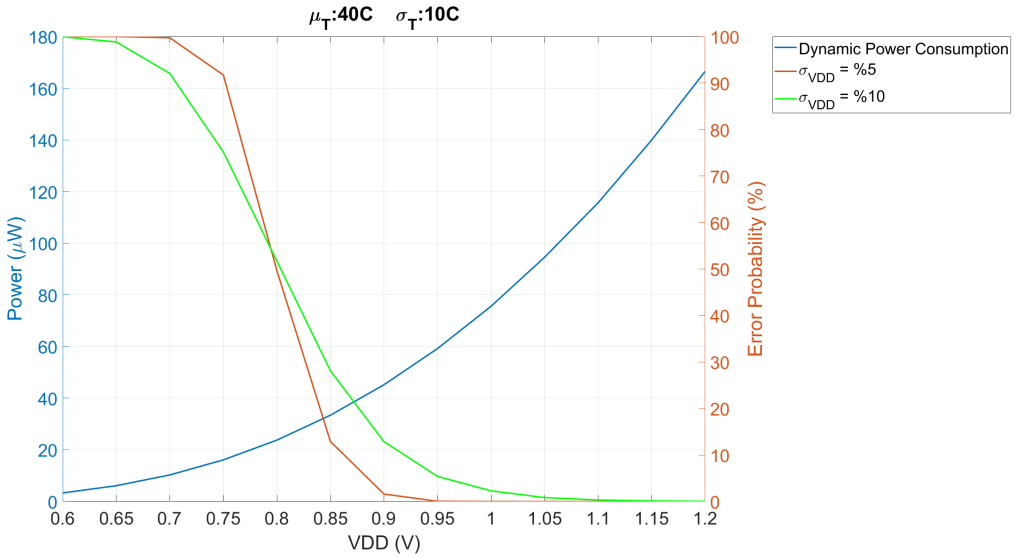

Figure 2.27 MAC Unit Error Probability vs Dynamic Power Consumption at Different Noise Level (1 Clock Period of Time)

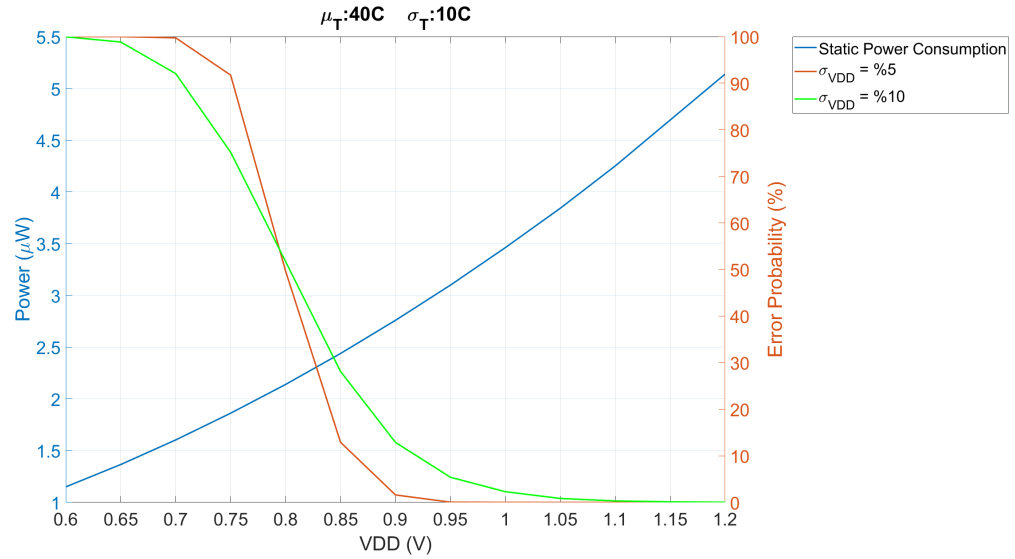

Figure 2.28 MAC Unit Error Probability vs Static Power Consumption at Different Noise Level (1 Clock Period of Time)

Dynamic power consumption reduces significantly with the voltage reduction whereas static power consumption is not reduced as much as dynamic power consumption in terms of ratio.

A study shows that, even in the range of 10^{-3} timing error rate leads to significant accuracy drop for DNN models (Jiao, Luo, Lin & Gupta (2017)). So, the main focus of probabilistic timing error model is low error rates. Therefore, when σ_{VDD} is %5, reducing voltage supply to $1.05V$ reduces power consumption approximately %43 with %0.0011 error probability of the MSB. When σ_{VDD} is %10, reducing voltage supply is not practical for the applications since circuit has error probability rate %0.758 already at nominal voltage $1.2V$. Depending on the DNN models or layers in the same DNN model, dynamic voltage scaling can be applied to reduce the power consumption. If a layer is robust to bit error probabilities, then voltage reduction can be applied for that layer. If the accuracy is of the layer is crucial, then supply voltage can be increased to reduce timing errors.

When green ($\sigma = \%10$) and red ($\sigma = \%10$) curves in Figure 2.27 are compared, the following conclusions are observed. At higher voltages, high voltage variations lead to increased error probabilities, since VDD has a higher likelihood of being lower than the nominal voltage. Voltage fluctuations can affect performance positively or negatively and that makes this trend reverse at lower voltages, since VDD has a higher likelihood of being higher than the nominal voltage. Probability of exceeding the nominal voltage for VDD is not realistic as voltage reduction case. Error probability calculation methodology could be updated based on the voltage level.

These results are accumulated error probability and as already discussed, PVT aware model error probability is independent from the input combinations. That means, actual error probability might lower then the analysis since critical paths might not triggered for all calculations.

3. DNN ACCELERATOR

3.1 Introduction

This chapter explains how to accelerate DNN operations, by examining the DNN architecture and showing small demonstrations. Then, designed prototype accelerator at 65nm CMOS technology, is explained by showing the details of the accelerator.

3.2 DNN Architectures and How to Accelerate

DNNs are composed of multiple layers that transform input data into meaningful outputs. A traditional fully connected neural network is that made up of layers of neurons depicted in Figure 3.1.

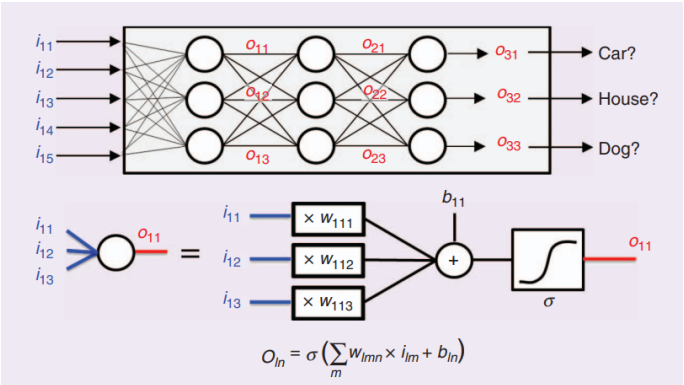

Figure 3.1 Fully Connected Neural Network (Verhelst & Moons, 2017)

These layers include the input layer, which receives the raw data just transfers to next layer. Hidden layers are utilized for processing the data through multiple neurons to extract features. Lastly output layer is used which produces the final prediction or classification. Each layer consists of nodes (neurons) connected to the next layer, with weights assigned to these connections that adjust during training to minimize error. An essential component of each neuron is the activation function, which introduces non-linearity into the network, allowing it to model complex patterns and relationships in the data. Common activation functions include Rectified Linear Unit (Relu), Sigmoid, and Tanh, each contributing to the network's ability to learn and represent complex data features. The complexity and depth of these layers and activation functions enable DNNs to perform effectively (Goodfellow et al. (2016)).

Convolutional Neural Networks (CNN) are extensively used for image-related AI tasks due to their unique architecture and ability to automatically and adaptively learn spatial hierarchies of features from input images. CNNs use filters (also

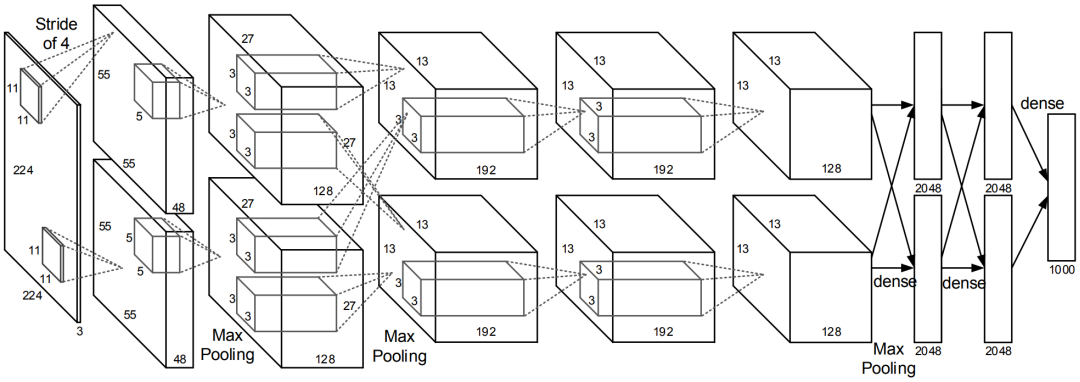

Figure 3.2 AlexNet Architecture (Krizhevsky et al., 2012a)

called kernels) that slide over the input image to detect patterns such as edges, textures, and more complex shapes. These filters are learned during training, allowing the network to adapt to the specific features of the training data. The same filter is used across different parts of the image, which reduces the number of parameters and helps the model generalize better. Figure 3.2 shows the architecture of the AlexNet. As depicted in the figure, AlexNet employs CNN and dense layers (fully connected layers) which requires extensive multiply and accumulate operations. Both CNN and fully connected layer operations can be accelerated with the accelerator along with similar mapping.

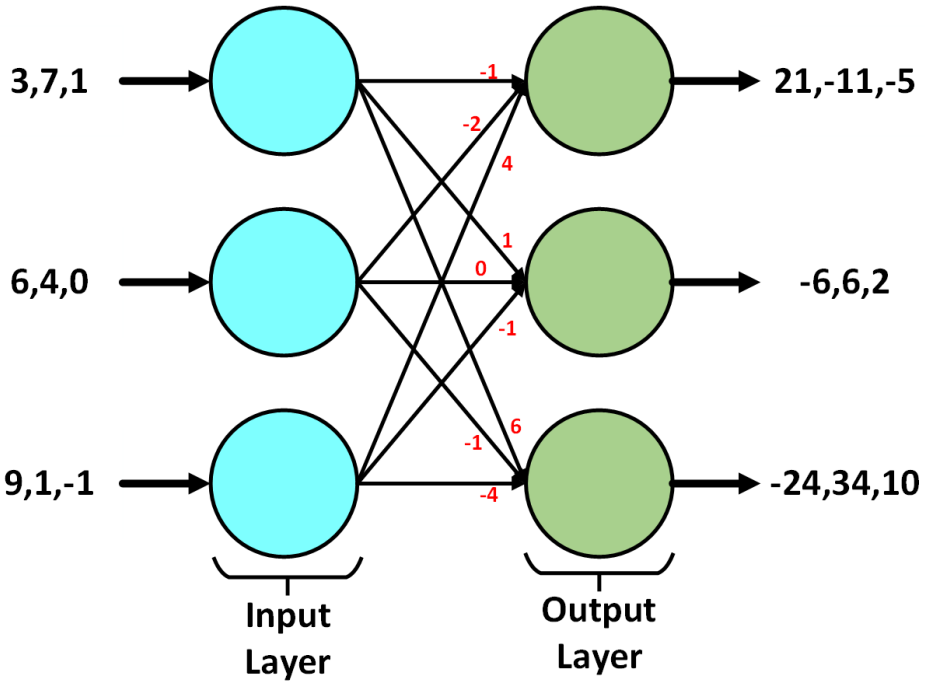

Figure 3.3 Fully Connected Layer

An example of fully connected layer which has 3 neurons is shown on Figure 3.3. Weights of the neurons are shown on the arrows in red color. It takes 3 different 3x1 sized input and its corresponding 3x3x1 sized output is shown.

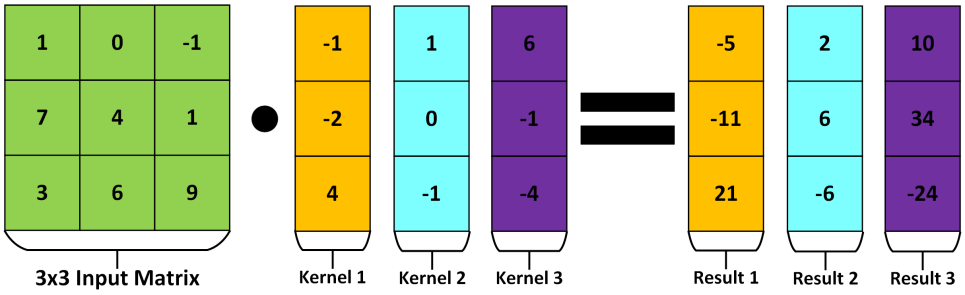

Figure 3.4 Convolution Neural Network Layers

CNN layer can be represented similar to fully connected layer. For instance, input matrix which has 3x3x1 dimension convolved with 3 kernels which their dimension size is 3x1. The resulted matrix has dimension 3x3x1 as shown on the Figure 3.4.

As depicted in both Figures 3.3 and 3.4, both layer have different input combinations (sizes), the same weight and result combinations. Both layers correspond to the same operations and its demonstration is shown Figure 3.5 to Figure 3.15. Yellow color is used to show any change in MAC unit or Register File. Activation transfer is shown as red color and intermediate result is shown with green color.

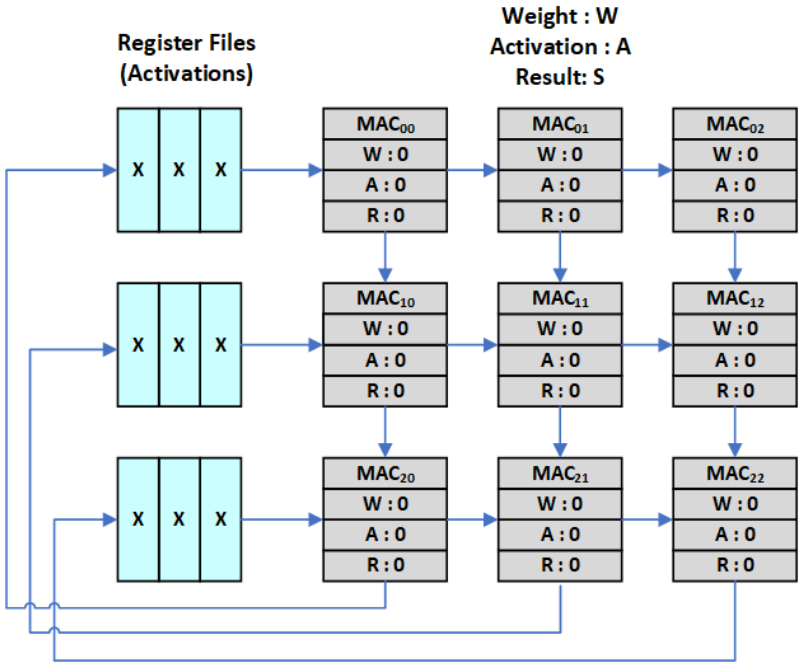

Figure 3.5 Accelerator, Memory and MAC Array Initial State

After reset signal, as shown Figure 3.5 internal Register Files are empty and their values are unknown, and all MAC Units' weight, activation and result registers are 0. Before starting the acceleration operation, activations are loaded to Register Files, weights are mapped to MAC Units.

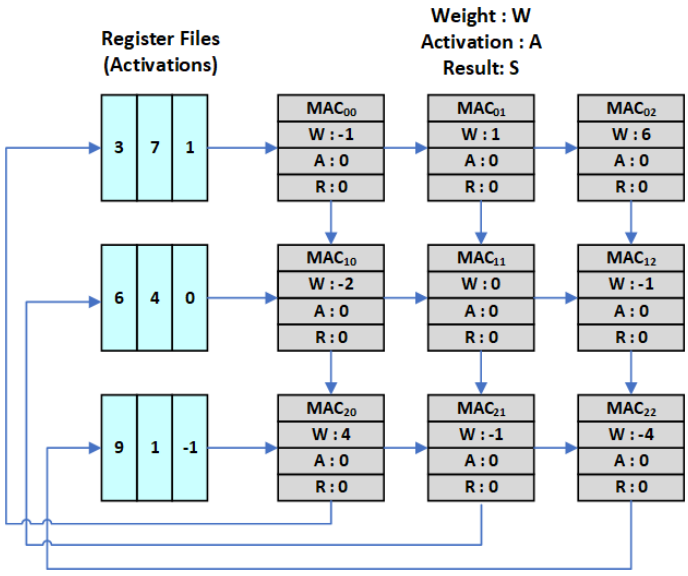

Figure 3.6 Step 1

After loading, the values are indicated in Figure 3.6.

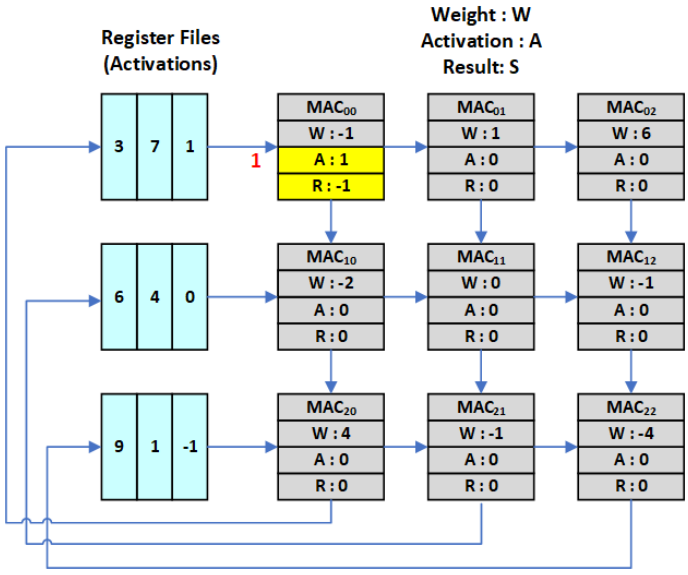

Figure 3.7 Step 2

In Step 2, first input comes from the memory for the left top MAC unit (MAC_{00}) and, it's activation is changed. Accumulated result is also changed as $1*(-1)+0=-1$ (since there is no intermediate result for the first row MAC Units, their input sum are 0).

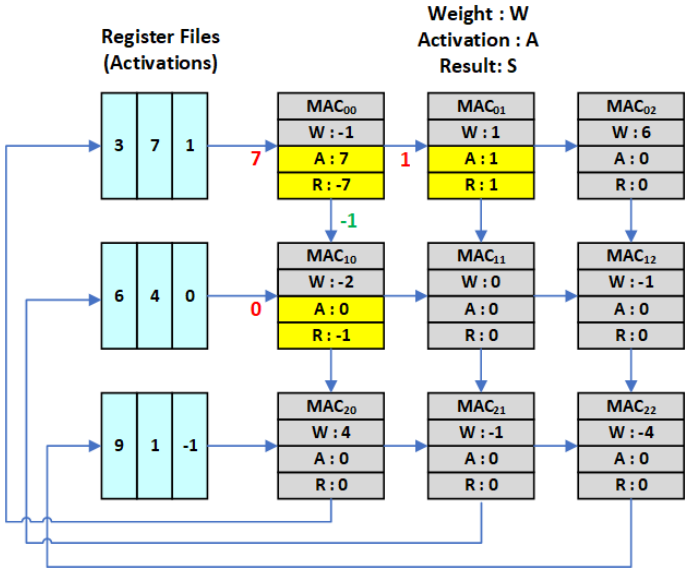

Figure 3.8 Step 3

In Step 3, second input of the MAC_{00} comes from memory, MAC_{01} receives its input from the MAC_{00}. Also MAC_{10} receives first input from the memory. Moreover, MAC_{10} receives intermediate result from the MAC_{00} as previously calculated and its accumulated result is $0*(-2)+(-1)=-1$.

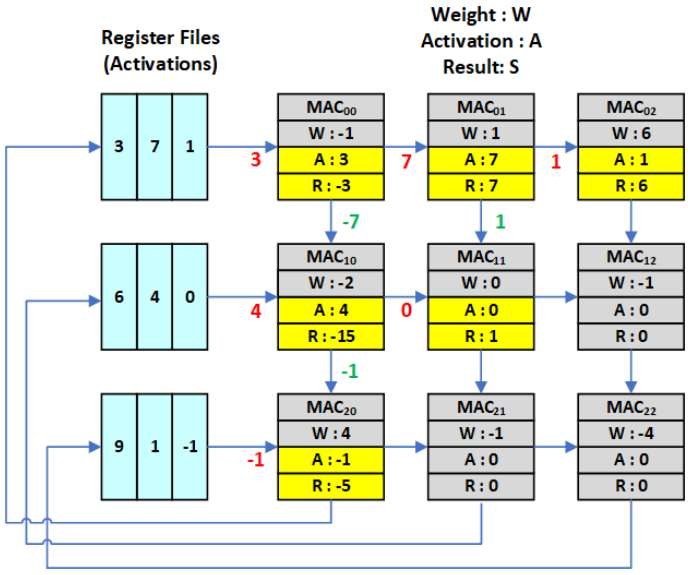

Figure 3.9 Step 4

In Step 4, last activation of the first memory is sent to MAC array. Read operation is complete for the first memory and it is ready to receive calculated results.

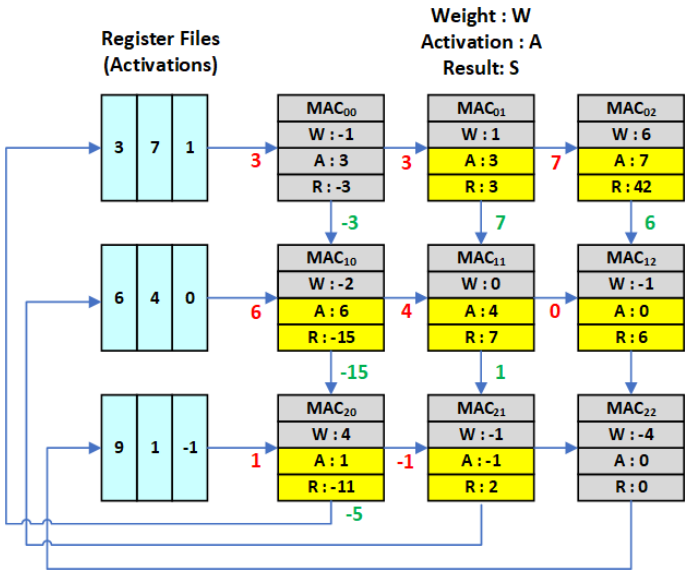

Figure 3.10 Step 5

In Step 5, first accumulated result is calculated as -5 by the first column of MAC Units.

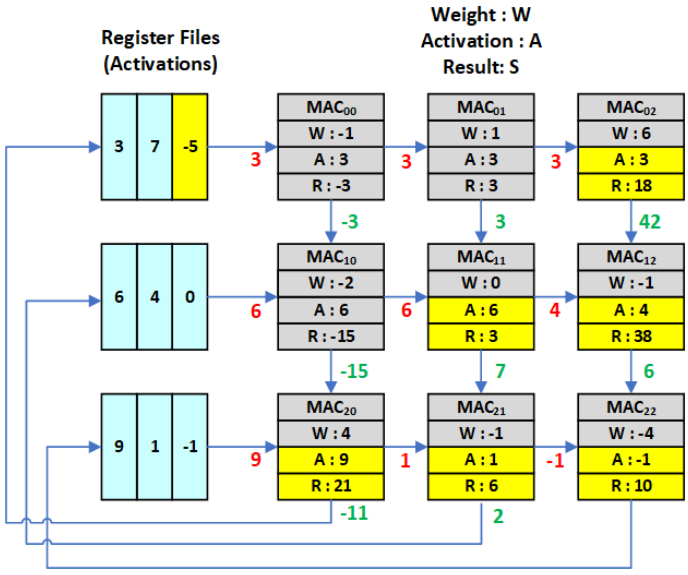

Figure 3.11 Step 6

In Step 6, first output of the first column is written back the memory. This flow continuous until all results are calculated.

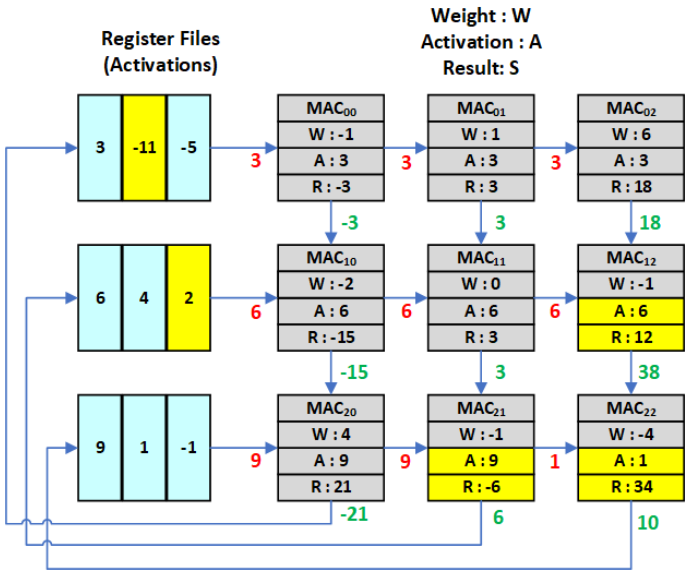

Figure 3.12 Step 7

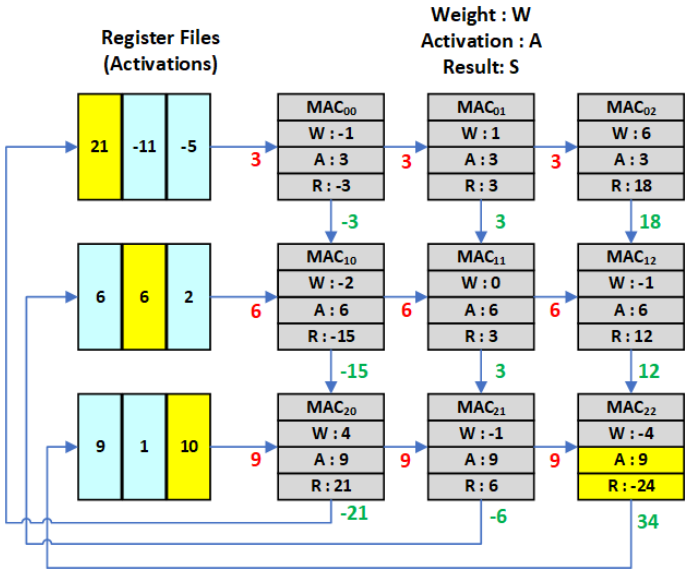

Figure 3.13 Step 8

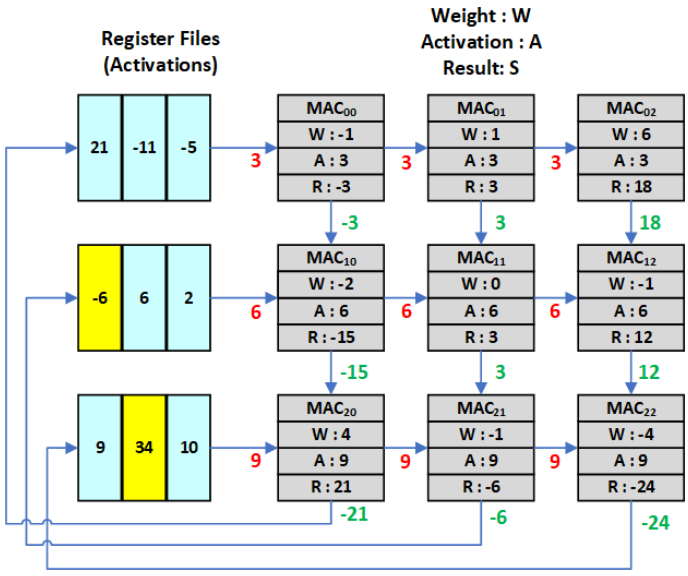

Figure 3.14 Step 9

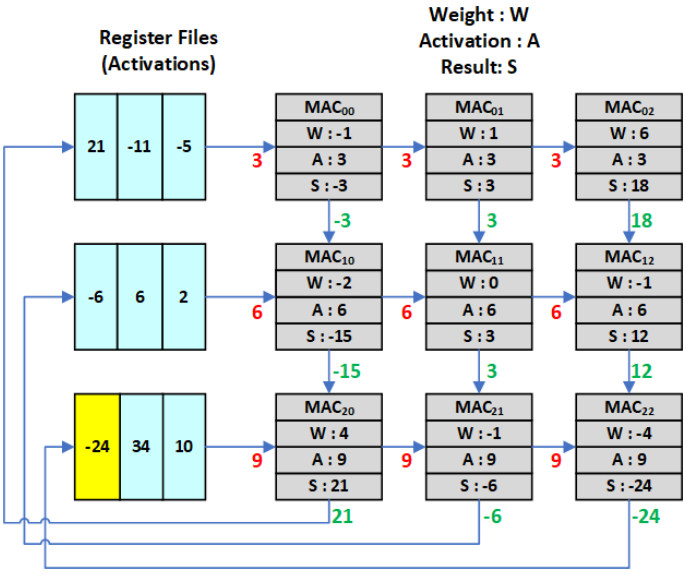

Figure 3.15 Accelerator, Memory and MAC Array Final State

In the final state, all outputs are written back to the on-chip memory and internal memory content is sent back to off-chip memory. Then, new activations are loaded to internal memory, and if necessary, new weights are mapped to MAC units.

Listing 3.1 Pseudo Code of NxN Dimension Matrix Multiplication

```
for(int e = 0; e < N; e++) {
    for(int d = 0; d < N; d++) {
        for(int a = 0; a < N; a++) {
            result[e][d] += A[e][a] * B[a][d];
        }
    }
}
```

NxN dimension MAC array accelerator requires 2x(NxN) read from Random Access Memory (RAM) for reading activations and weights, NxN write access to RAM for writing back the results. That means accelerator has O(N^2) RAM access complexity. Roughly 3xN clock cycle is required for calculation of all the results which means it has O(N) time complexity for all calculations.

Software code is given above for matrix multiplication. On the other hand, NxN sized matrix multiplication operation requires 2x(NxNxN) read from memory and NxN write back to memory on CPU implementation. Both time and RAM access complexities are O(N^3).

GPU and FPGA are not included to Table 3.1 since both can be used to achieve the same time and RAM access complexities as ASIC accelerator. However, assuming

	Calculation Time Complexity	RAM Access Complexity
MAC Array Accelerator	O(N)	O(N^2)
CPU	O(N^3)	O(N^3)

Table 3.1 CPU vs MAC Array Accelerator Comparison

that they are manufactured with the same technology, with the proper implementation in architectural level, ASIC has higher efficiency in terms of power, area and speed.

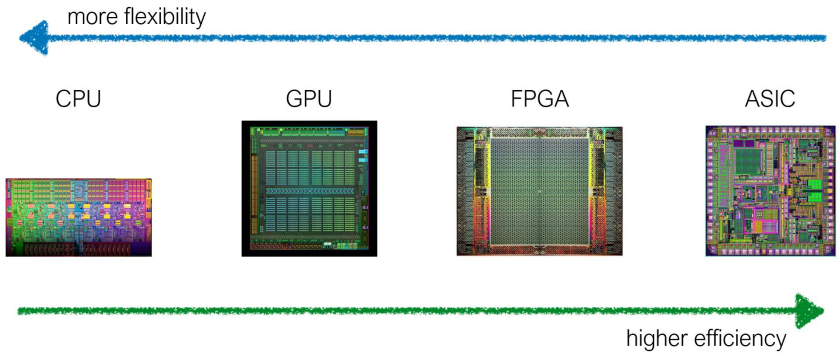

Figure 3.16 CPU, GPU, FPGA, ASIC (Overflow, 2018)

3.3 16x16 Size Systolic Array

Designed DNN accelerator consists of 16x16 sized systolic MAC array, 16x40 byte on-chip Register File, and the controller block to control accelerator and memory. Memory and controller blocks are powered with nominal voltage level which 1.2V by digital power pads. MAC array part is powered by analog power pads, that means its voltage level can be adjustable (mostly lower than 1.2V) to demonstrate the proposed PVT aware model. In the literature, there are proposed ASIC accelerators work at different frequencies up to 1Ghz (Dhilleswararao, Boppu & others (2022)). Since this study aims to IoT devices due to low power requirement, this accelerator is designed to work at 200Mhz. Detail of each block is explained in section 3.4.

Various Electronic Design Automation (EDA) tools were employed to develop the accelerator. Vivado was used for HDL design and behavioral verification. Afterwards, Synopsys Design Compiler was used for synthesizing the netlist at 200Mhz. Each block was synthesized separately since synthesizing entire design as a whole, takes huge amount of time. Then Cadence Innovus tool was used for Clock Tree

Synthesis (CTS), placing and routing. Innovus's DRC, LVS and antenna check was used to check the design. Design was imported to Cadence Virtuoso and lastly Calibre tool was employed to perform detailed DRC and LVS checks. The chip was manufactured. However, printed circuit board (PCB) for the chip has not been finalized. The planned setup is going to be explained at section 3.5.

3.3.1 Development Process of Accelerator

Initially various architectures were discussed. The computations can be mapped onto a systolic array, with various options available Chen et al. (2017). These different mappings, known as dataflows, can be categorized into weight stationary, output stationary, and input stationary based on the underlying data reuse patterns. The order of input feature maps and weights being fed into the array, and how intermediate partial sums are stored and utilized, is determined by these dataflows, as explained further.

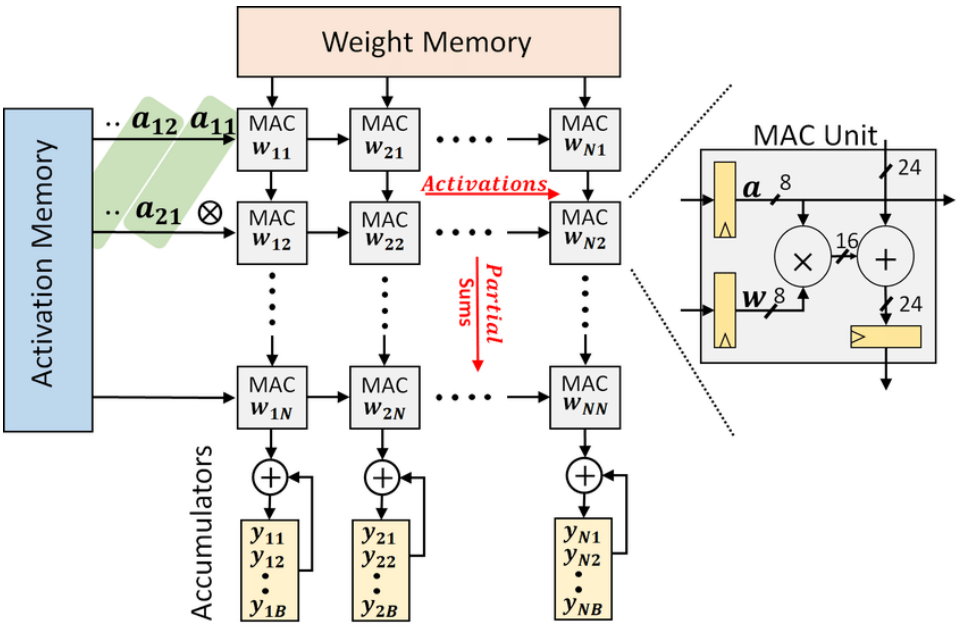

Figure 3.17 (Zhang et al., 2018)

- Weight stationary dataflow refers to a mapping where weights are deployed within the systolic array while inputs flow into the array in each cycle, multiplied by their corresponding weights. The partial sums are accumulated across multiple MAC units within a specific column of the systolic array over several cycles. After completing all computations for a particular set of weights, the mapping is repeated for

the subsequent set of weights and so on. In this approach, inputs change cycle by cycle while weights remain stationary in the array.

- Input stationary dataflow shares similarities with weight stationary dataflow, except that the input feature map which is mapped onto a systolic array and remains fixed while the weights are fed into the systolic array.

- In the context of output stationary dataflow, each unit calculates each pixel within the output feature map. This mapping involves streaming each necessary input and weight for a particular output pixel per cycle, while accumulating them in the corresponding MAC Unit. Once an activation is generated, it's moved to memory, and the MAC Unit begins computing the next output pixel. The process continues in this manner.

Weight stationary was chosen among the 3 architectures due to simplicity of the acquiring the results from the array (there is no real difference input and weight stationary architecture in terms of hardware implementation). A small prototype is being used for to verify the probabilistic model. Architecture that shown in Figure 3.17, was implemented using Verilog HDL. Parametric implementation was preferred for future update of the array size, so that weight and activation bit sizes, MAC array size (N must be power of 2 and array size must be NxN) can be changed via parameters. This Verilog implementation allows functional test on FPGA before moving on to ASIC implementation with desired parameters.

Since we wanted to support different types of DNN models (even different layers in the same model might vary in terms of bit representation), fixed point representation is not used on the accelerator. Moreover, it does not affect the result in terms of bit representation for MAC operations (bias is not included to outputs immediately since bias value has fixed point representation whereas outputs don't have). An example is shown in 3.1 and 3.2. It multiplies 3 bit numbers and adds it to 8-bit number. As it can be seen, after quantization different real numbers can have the same bit representation for accelerator ("." is used for separating the decimal part).

(3.1)

Weight:2 bit integer, 1 bit decimal

$2.5 = 1*2^1 + 0*2^0 + 1*2^{-1}$ (10.1)

Activation:3 bit integer, 0 bit decimal

$3 = 0*2^2 + 1*2^1 + 1*2^0$ (011)

Intermediate product:7 bit integer, 1 bit decimal

$49 = 0*2^6 + 1*2^5 + 1*2^4 + 0*2^3 + 0*2^2 + 0*2^1 + 1*2^0 + 0*2^{-1}$ (011 0001.0)

Result:7 bit integer, 1 bit decimal

$2.5*3 + 49 = 56.5$

$10.1*011 + 011\ 0001.0 = 011\ 1000.1$

(3.2)

Weight:0 bit integer, 3 bit decimal

$0.625 = 1*2^{-1} + 0*2^{-2} + 1*2^{-3}$ (.101)

Activation:1 bit integer, 2 bit decimal

$0.75 = 0*2^0 + 1*2^{-1} + 1*2^{-2}$ (0.11)

Intermediate product:3 bit integer, 5 bit decimal

$3.0625 = 0*2^2 + 1*2^1 + 1*2^0 + 0*2^{-1} + 0*2^{-2} + 0*2^{-3} + 1*2^{-4} + 0*2^{-5}$ (011.0001 0)

Result:3 bit integer, 5 bit decimal

$0.625*0.75 + 3.0625 = 3.53125$

$.101*0.11 + 011.0001\ 0 = 011.1000\ 1$

However, to add bias properly, accelerator has to know which bits represent what. For example adding bias as 12.625 (**in binary 01100.101, 5 bit integer, 3 bit decimal**) result in different accuracy loss for both results even the same bit size is used for representation.

$56.5 + 12.625 = 69.125$ (100 0101.001) $correct\,result$

$011\ 1000.1 + 0\ 1100.101 = 100\ 0101.0$ $actual\,result$

$accuracy\,loss$: 0.125

(3.3)

$3.53125 + 12.625 = 16.15625$ (1 0000.00101) $correct\,result$

$011.1000\ 1 + 0\ 1100.101 = 1\ 0000.001$ $actual\,result$

$accuracy\,loss$: 0.03125

Since most significant bit MSB and least significant bit LSB of 56.5 represents 2^6, 2^{-1} respectively (precision is 2^{-1}), by employing 8-bit adder, bias value loses 2 LSBs and 0.125 accuracy is lost during bias addition. On the other hand, for the second addition, MSB and LSB are 2^4, 2^{-3} respectively (precision is 2^{-3}). Calculated result loses 2 LSBs and accuracy loss is 0.03125. As it can be seen, for bias addition, circuit has to keep track of the bit representation. So, the information about what the MSB of the activation, weight and bias values represent should be provided to the accelerator.

3.3.2 Activation Functions

The activation functions are integral in introducing non-linearity into the neural network, empowering it to learn complex relationships between features. Common activation functions include Rectified Linear Unit (Relu), Sigmoid, and Tanh. For instance, Relu substitutes all negative values with zero, facilitating effective modeling of non-linear patterns within the network. Without activation functions, the network would behave like a linear model, limiting its capacity to capture sophisticated data patterns. The introduction of activation functions enriches network functionality and empowers it to learn more complex, non-linear relationships between input features (LeCun et al. (2015)).

Activation functions Sigmoid, Hyperbolic Tangent, and Relu are implemented. To facilitate implementation on ASIC, piecewise linear approximation approach is utilized for complex functions, ensuring efficient hardware usage (Tsmots, Skorokhoda & Rabyk (2019)).

$$sgm(x) = \frac{1}{1+e^{-x}} \tag{3.4}$$

$$tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \tag{3.5}$$

Sigmoid and Hyperbolic Tangent functions are shown in equation 3.4, and 3.5 respectively. The points selected to divide the Sigmoid and Hyperbolic Tangent function into pieces, are shown in Figure 3.18, 3.20. Sigmoid and Hyperbolic Tangent are divided into 18 lines ($y = a * x + b$) and their formulas are shown in Table 3.19, 3.21

respectively.

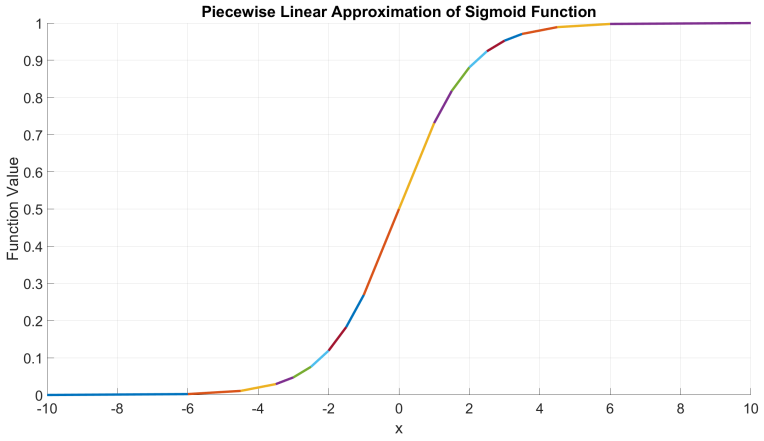

Figure 3.18 Sigmoid Function, Used Points for Piece-wise Linear Division

Area	a	b	Representation of "a" in 8 Bits	Representation of "b" in 11 Bits
X > 6	0	255/256	".00000000"	"000.11111111"
6 > X > 4.5	1/256	247/256	".00000001"	"000.11110111"
4.5 > X > 3.5	5/256	232/256	".00000101"	"000.11101000"
3.5 > X > 3	9/256	216/256	".00001001"	"000.11011000"
3 > X > 2.5	15/256	200/256	".00001111"	"000.11001000"
2.5 > X > 2	22/256	181/256	".00010110"	"000.10110101"
2 > X > 1.5	32/256	161/256	".00100000"	"000.10100001"
1.5 > X > 1	44/256	143/256	".00101100"	"000.10001111"
1 > X > 0	59/256	128/256	".00111011"	"000.10000000"
0 > X > -1	59/256	128/256	".00111011"	"000.10000000"
-1 > X > -1.5	44/256	143/256	".00101100"	"000.10001111"
-1.5 > X > -2	32/256	161/256	".00100000"	"000.10100001"
-2 > X > -2.5	22/256	181/256	".00010110"	"000.10110101"
-2.5 > X > -3	15/256	200/256	".00001111"	"000.11001000"
-3 > X > -3.5	9/256	216/256	".00001001"	"000.11011000"
-3.5 > X > -4.5	5/256	232/256	".00000101"	"000.11101000"
-4.5 > X > -6	1/256	247/256	".00000001"	"000.11110111"
-6 > X	0	255/256	".00000000"	"000.11111111"

Figure 3.19 Sigmoid Function, Line Coefficients

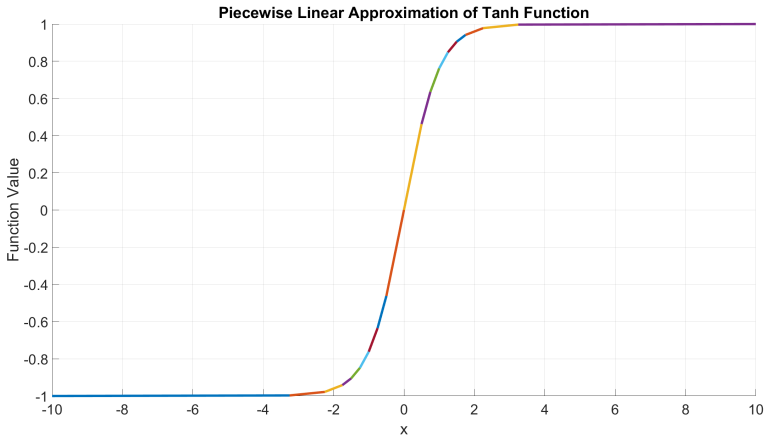

Figure 3.20 Hyperbolic Tangent Function, Used Points for Piece-wise Linear Division

Hyperbolic Tangent Function	a	b	Representation of "a" in 8 Bits	Representation of "b" in 11 Bits
X > 3.25	0	255/256	".00000000"	"000.11111111"
3.25 > X > 2.25	5/256	239/256	".00000101"	"000.11101111"
2.25 > X > 1.75	19/256	208/256	".00010011"	"000.11010000"
1.75 > X > 1.5	37/256	176/256	".00100101"	"000.10110000"
1.5 > X > 1.25	58/256	144/256	".00111010"	"000.10010000"
1.25 > X > 1	89/256	106/256	".00111011"	"000.01101010"
1 > X > 0.75	129/256	65/256	".10000001"	"000.01000001"
0.75 > X > 0.5	177/256	30/256	".10110001"	"000.00011110"
0.5 > X > 0	237/256	0	".11101101"	"000.00000000"
0 > X > -0.5	237/256	0	".11101101"	"000.00000000"
-0.5 > X > -0.75	177/256	-30/256	".10110001"	"111.11100010"
-0.75 > X > -1	129/256	-65/256	".10000001"	"111.10111111"
-1 > X > -1.25	89/256	-106/256	".00111011"	"111.10010110"
-1.25 > X > -1.5	58/256	-144/256	".00111010"	"111.01110000"
-1.5 > X > -1.75	37/256	-176/256	".00100101"	"111.01010000"
-1.75 > X > -2.25	19/256	-208/256	".00010011"	"111.00110000"
-2.25 > X > -3.25	5/256	-239/256	".00000101"	"111.00010001"
-3.25 > X	0	-255/256	".00000000"	"111.00000001"

Figure 3.21 Hyperbolic Tangent Function, Line Coefficients

$$Relu(x) = max(0, x) \tag{3.6}$$

The most used activation function in DNNs is Relu, which eliminates the negative results of a given input.

User can choose to get results directly or after passing through an activation function.

3.3.3 Final Architecture

For the proof of concept, small size MAC array is chosen to be implemented to reduce the chip area, eventually manufacturing cost. Since MAC array size is reduced to 16x16, large DNN model layers have to be divided into multiple blocks. All of the necessary weights and activations couldn't be kept in the internal SRAMs. Memory size is also reduced along with array size since most of the chip area was dominated by the SRAMs. There were 3 distinct memory blocks which are activation, weight and result memory as shown Figure 3.17 at the beginning. However since we had to divide the layer into parts, memory size should be compatible with the array size. There was no need to use excess memory. So weight and result memory are discarded from the accelerator and there is only one memory block which consist of 16 distinct Register Files (RF). Initially, activations are written to the RFs, weights are directly mapped to MAC units, and finally, results are stored back into the register files. Then the results are sent back to off-chip. Activation functions and bias addition are removed to make accelerator simpler and smaller. The final architecture is depicted in the Figure 3.22.

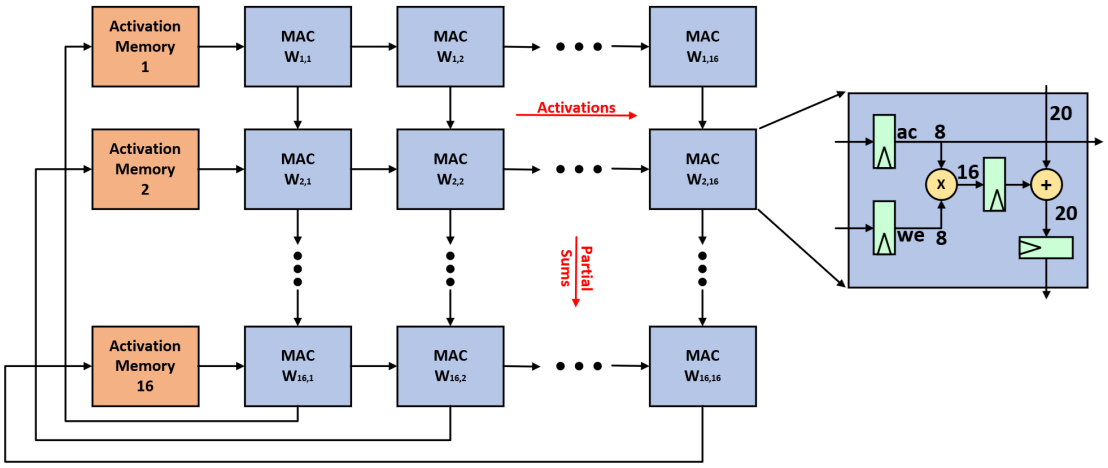

Figure 3.22 Final Block Level Schematic of 16x16 MAC Array

Initially 10-bit quantization was planned to be implemented however, 8-bit quantization was chosen to operate, since it provides good balance between accuracy and speed Jacob, Kligys, Chen & others (2018). On the other hand, pipeline register was utilized between multiplier and adder to operate at faster clock frequency. For the MAC Unit, Ripple Carry Adder and Array Multiplier were deployed due to simple implementation for parametric design to make a change if necessary. Schematics are shown in Figures 3.23, 3.24 respectively.

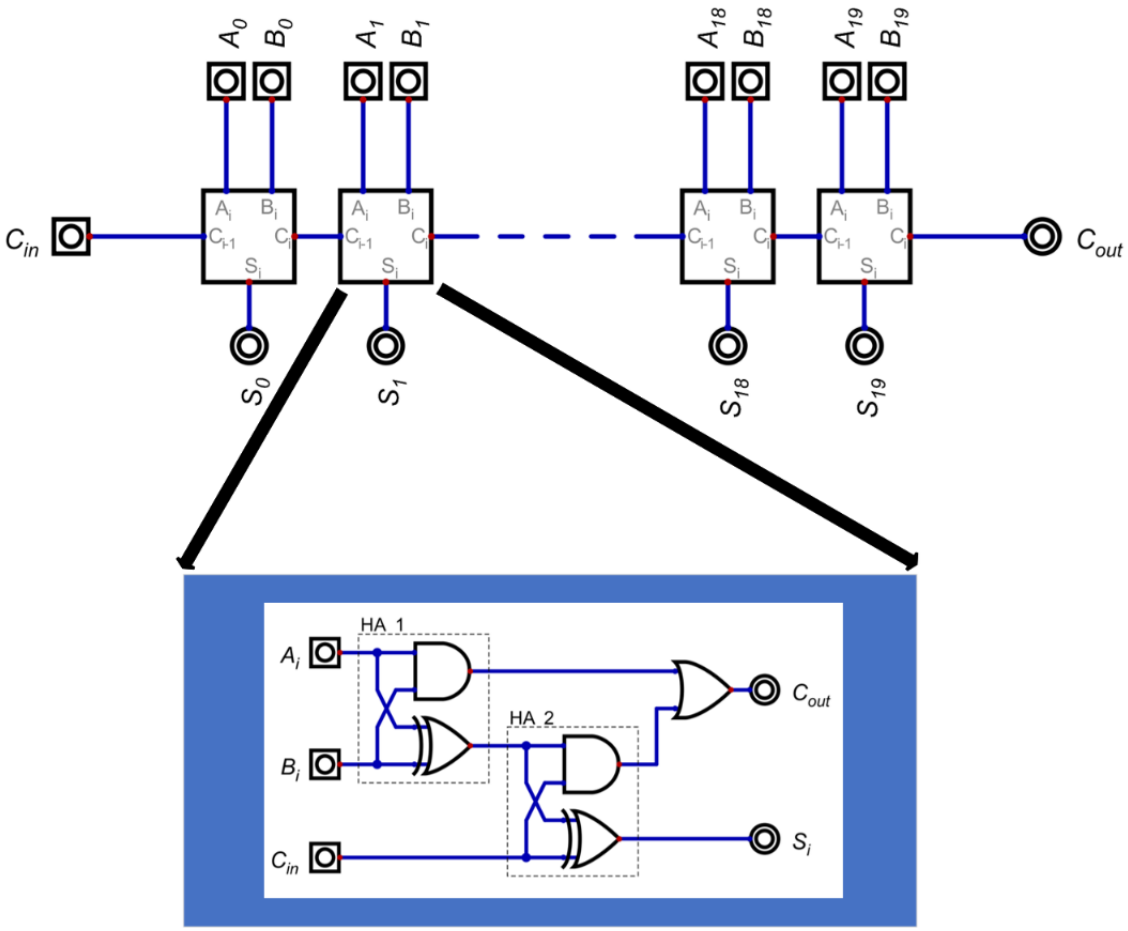

Figure 3.23 20-Bit Ripple Carry Adder Schematic

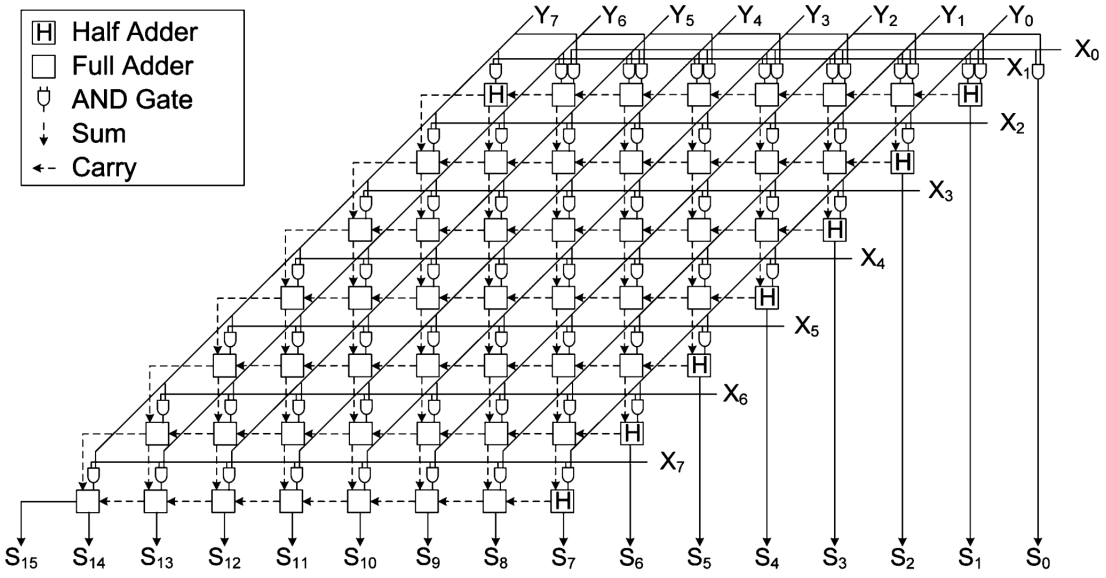

Figure 3.24 8-Bit Array Multiplier (Själander & Larsson-Edefors, 2009)

3.4 Accelerator Modules

This section examines each module of the accelerator. Top-down approach is used to explain the modules. Top view of the implemented accelerator is shown in Figure 3.25.

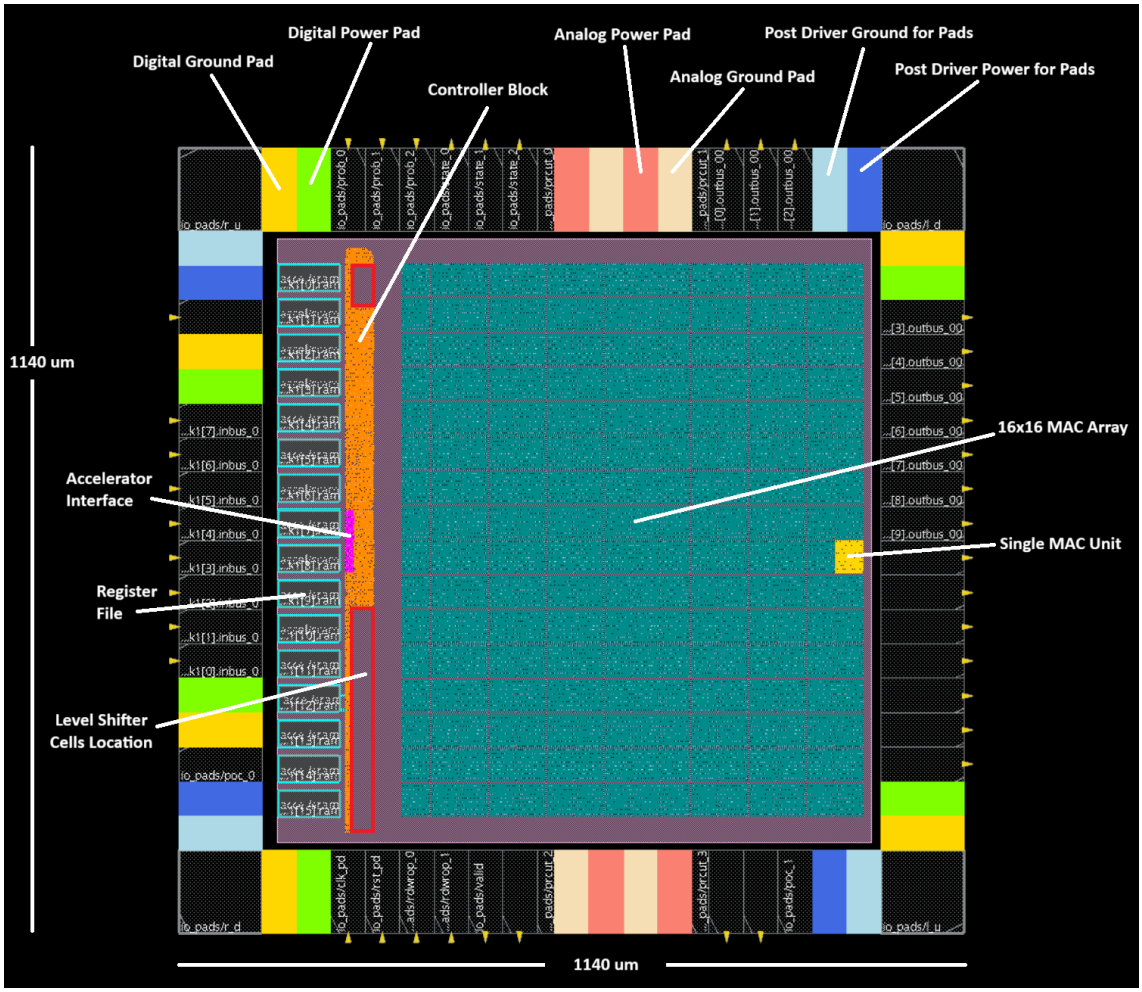

Figure 3.25 Top View of DNN Accelerator
$x = 1140\mu m$, $y = 1140\mu m$

3.4.1 DNN Accelerator

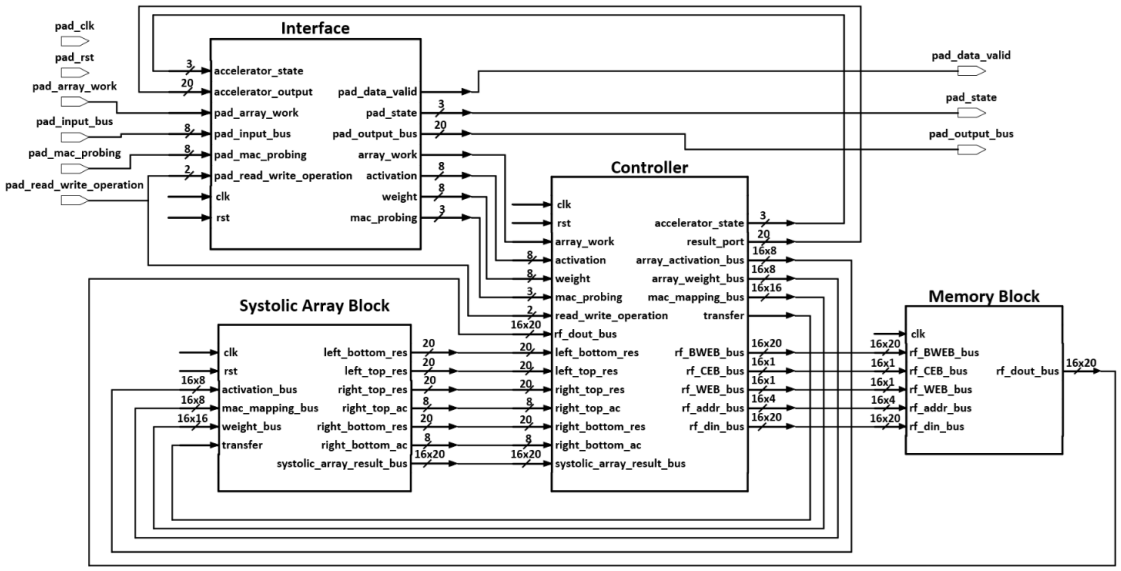

Figure 3.26 Schematic of DNN Accelerator

Chip is divided into 2 power domains which are digital and analog powered parts. The digital powered part consist of memory blocks, controller and accelerator interface block. The analog powered part includes only systolic MAC array.

Red boxes indicate low to high level shifter cells in Figure 3.25. Since we want to reduce voltage of systolic array part to observe the impact on the results, to be able to write the results back to memory properly, level shifter cells have been utilized. High to low level shifter cells have been also deployed in analog region, but since they are so small in terms of area, they are not indicated.

The orange and purple region indicate controller block and accelerator interface respectively in Figure 3.25. Register Files are being seen in the blue box as distinct blocks. Also, each MAC unit is being seen in the turquoise region. MAC units are identical and one of which is indicated as yellow.

Overall chip size increased with adding bond pads and total area was $1.31cm\ x\ 1.31cm = 1.7161cm^2$.

In total, 5 digital power pads, 5 digital ground pads, 4 analog power pads, 4 analog ground pads have been deployed in the design for powering the chip. 4 post driver voltage supply pads and 4 post driver ground pads have been used to feed the I/O pads. Analog pads have been isolated via power-cut pads, so the chip has 3 distinct islands in terms of pad connections. Each digital pad island requires power on

control (POC) pad to protect the chip from electrostatic discharge (ESD). Analog pads have an internal circuit that chip is protected from ESD. The rest are the I/O pads. In total 40 I/O pads, 26 power pads, have been used.

Innovus reports that accelerator consumes approximately $111\,mW$ power when both digital power supply (VDD) and analog power supply (AVDD) set to 1.2V.

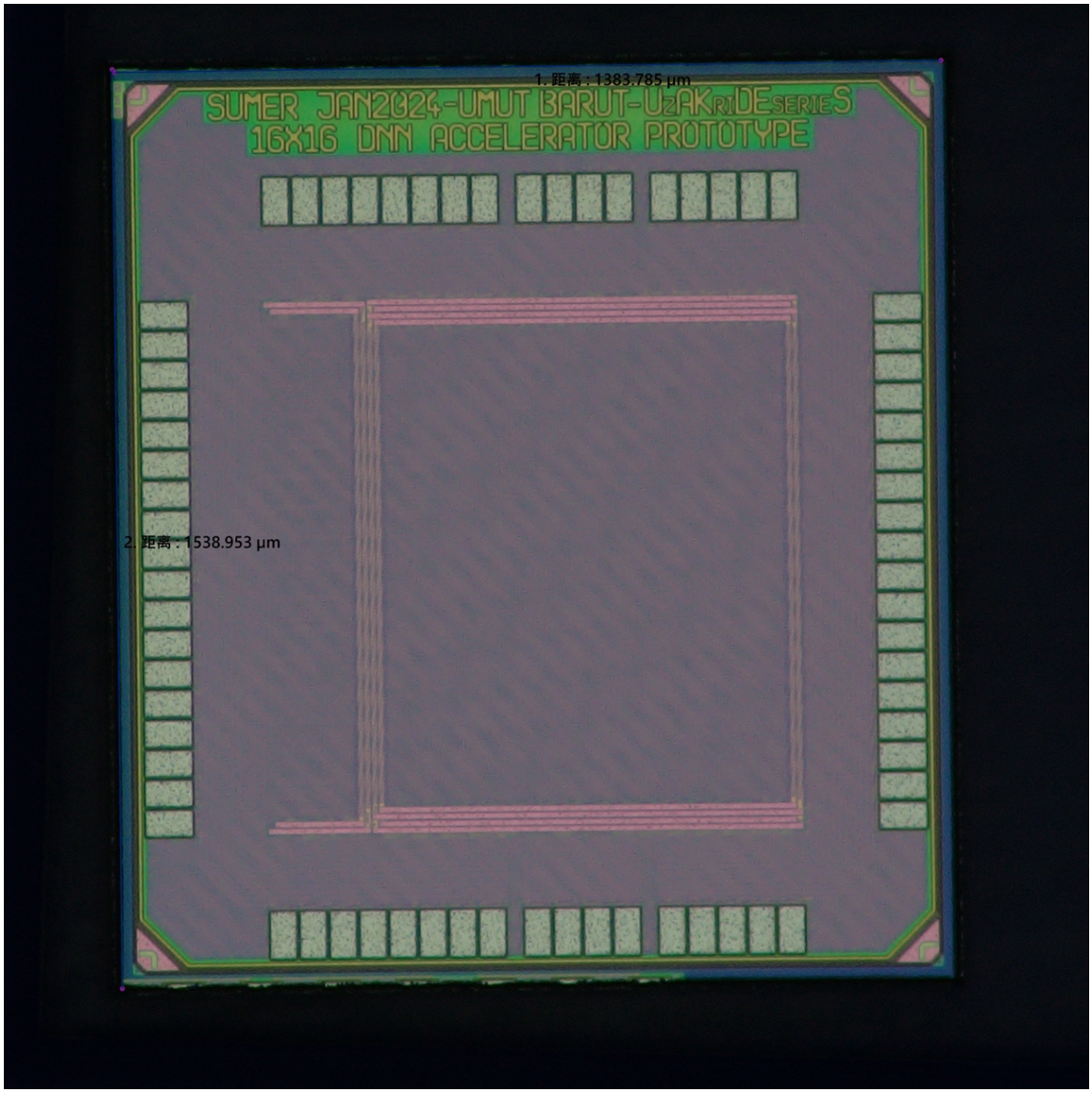

Figure 3.27 Top View of DNN Accelerator Under Microscope
$x = 1384\mu m$, $y = 1539\mu m$

The final design under microscope is shown in Figure 3.27. In total area is $2.13cm^2$.

3.4.2 16x16 Systolic Array

As depicted in the Schematic 3.26, systolic array takes inputs as activation bus, weight bus, mac mapping bus and lastly transfer signal from the controller.

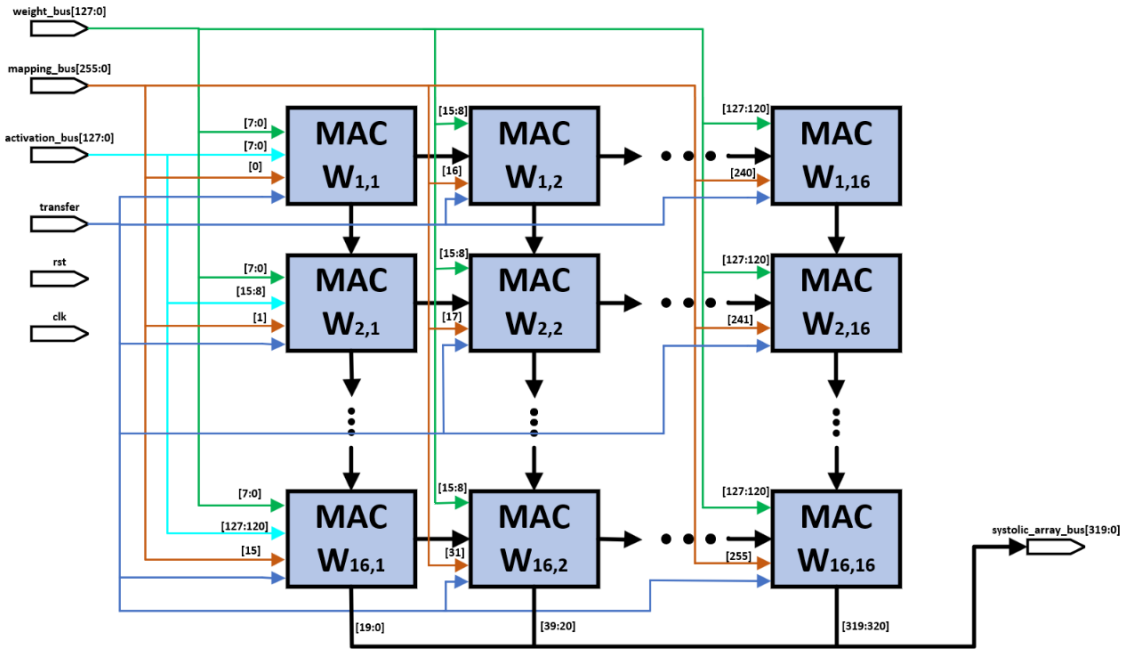

Figure 3.28 Systolic MAC Array

Activation bus connected to first column of the MAC array as shown in Figure 3.28. Each MAC unit has the "mapping" signal that enables keeping the weight in the register. So, in total 256 bit signal distributed among all MAC units. Weight bus distributed among the columns so that each column of MAC units has connection with the bus however they can't keep the weight unless own mapping signal is "1".

For simplicity, output signals are not shown except the "systolic_array_result_bus". Other output signals that end with "_res" and "_ac" are shown in Figure 3.26, sample the corresponding MAC unit's result and activation respectively. They have been utilized for debugging purposes.

3.4.3 MAC Unit

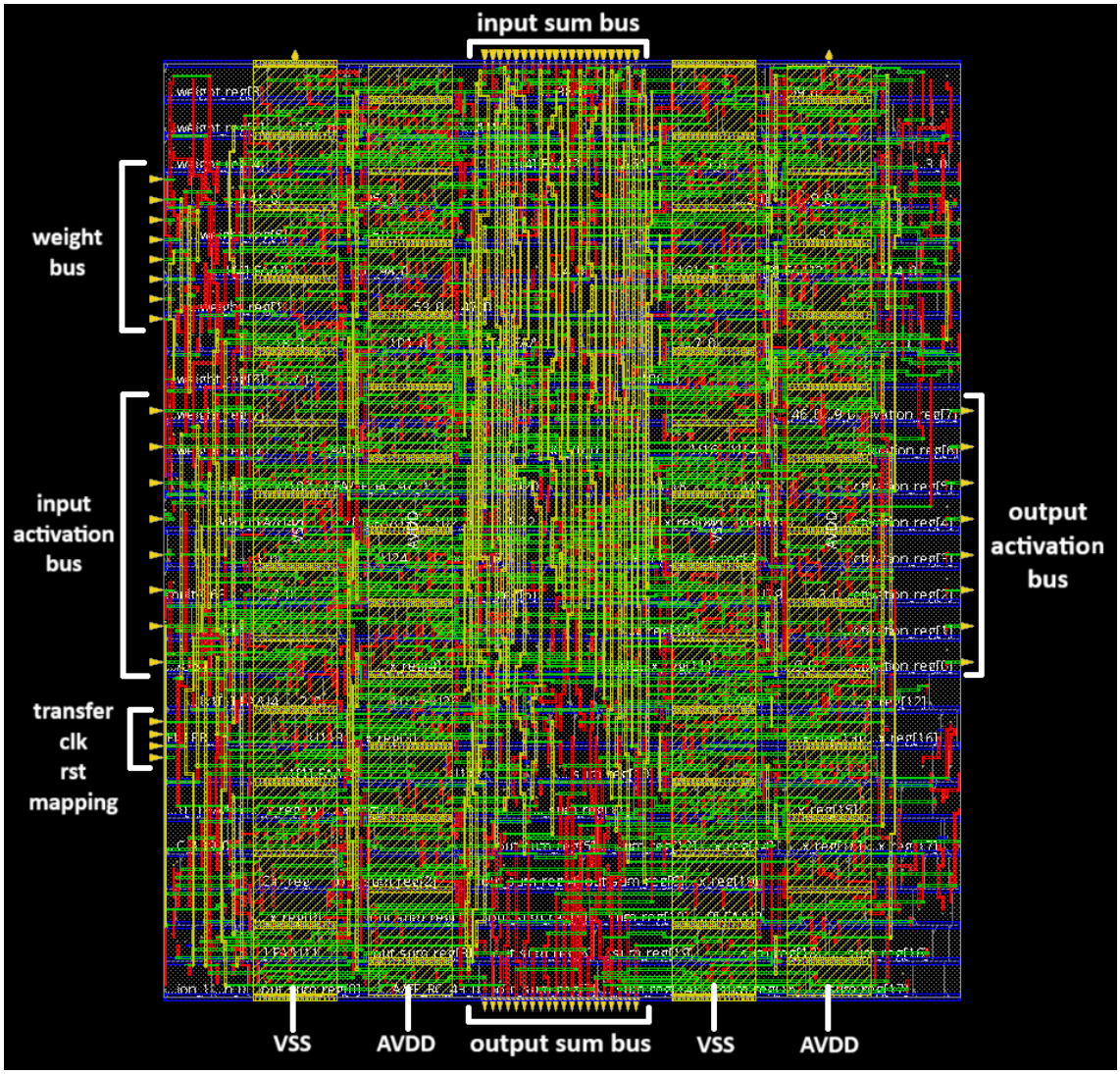

Figure 3.29 Top View of MAC Unit ($x = 40\mu m$, $y = 46.8\mu m$)

As shown in Figure 3.29, MAC unit does not have power ring itself since it makes complicated the placing and routing of MAC array. Instead of individual power ring, MAC array has the power ring and MAC units are fed with stripes. Each MAC unit has 2 pair of AVDD and VSS.

MAC array requires buffers to carry the signals further and CTS to balance the clock skew. In the top hierarchy (systolic array 3.28), buffers and MAC array share the same power rails so that MAC unit's vertical dimension has to be multiple of the height of standard cells which is $1.8\,\mu m$. Overall, MAC unit's dimension is $40\,\mu m\ x\ 46.8\,\mu m$.

Innovus reports that MAC unit's power consumption is approximately $91\,\mu W$.

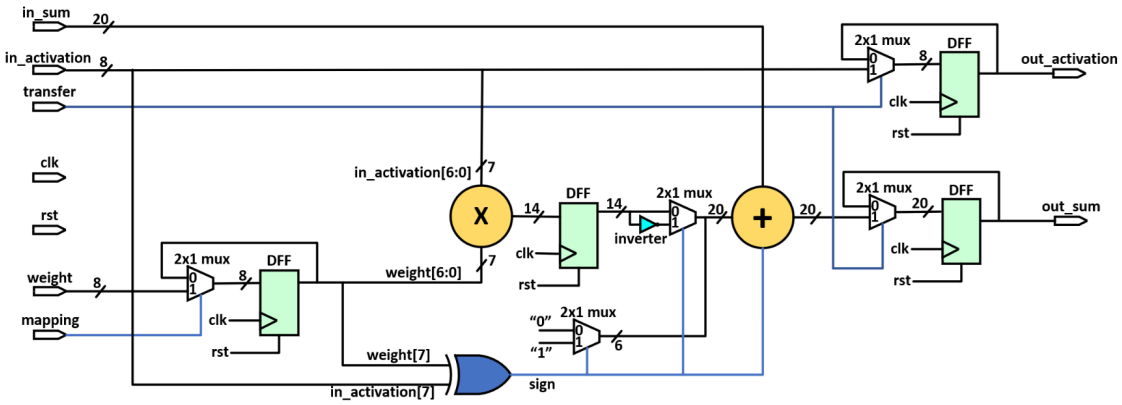

Figure 3.30 Schematic of MAC Unit

The block diagram of the MAC unit is shown in Figure 3.30. Since activations and weights can be negative, instead of representing them in 1's or 2's complement notation, the following notation is used.

- Most significant bit represents sign of the number. Logic "1" represents negative, logic "0" represents positive.
- Rest 7 bit represents the unsigned version of the number. For the adder, 2's complement representation was used to add the negative and positive results easily.
- In this representation, range of $[-127 : 127]$ is representable for activations and weights. In 2's complement notation, additionally -128 is representable and if it is equals to -128, basically -127 is used. By doing that, multiplication becomes easier since it does not require 2's complement to unsigned conversion. Thus, additionally the area was saved.
- Sign is determined by taking XOR of both MSB of activation and weight.
-Multiplication result is padded with respect to sign value. It padded with "000000" or "111111" when sign logic is "0" or "1" respectively. Additionally, if the sign logic is "1", inverted multiplication result is selected, and C_{in} also becomes "1" for addition. Thus 2's complement conversion for the multiplication result and summation with the previous result can easily be done in the same adder.

An example is demonstrated in Equation 3.7 as decimal and 3.8 as binary. "A,B" represents concatenation operation. Variable "adder_x" represents adder's second input.

$$
\begin{aligned}
&activation = 101 \\
&weight = -117 \\
&sign\,(Cin) = 1 \\
&multiple\_result = 11817 \\
&adder\_x = -11818 \\
&out\_result = in\_sum + adder\_x + sign \\
&in\_sum = -27507 \\
&out\_result = -39324
\end{aligned}
\tag{3.7}
$$

$$
\begin{aligned}
&activation = 01100101 \\
&weight = 11110101 \\
&sign\,(Cin) = 1 \\
&multiple\_result = 10\ 1110\ 0010\ 1001 \\
&adder\_x = \{11\ 1111, 01\ 0001\ 1101\ 0110\} \\
&out\_result = in\_sum + adder\_x + sign \\
&in\_sum = 1111\ 1001\ 0100\ 1000\ 1101 \\
&out\_result = 1111\ 0110\ 0110\ 0110\ 0100
\end{aligned}
\tag{3.8}
$$

3.4.4 Accelerator Interface

The Accelerator Interface's schematic view was not given since it is complicated. In this part, Accelerator Interface's purpose is explained. I/Os of the block is depicted in Figure 3.26.

- It is kind of a bridge between on-chip and off-chip. Signals that come from off-chip are registered and sent to Controller block. "clk", "rst" and "read_write _operation" signals are used without registered and they are sent to all necessary blocks.
- Activations and weights are given to accelerator via "input_bus" and Interface connects "input_bus" to "activation" or "weight" according to "read_write _operation" signal.
- Output signals are registered and sent to I/O pads.

3.4.5 Accelerator Controller

Accelerator Controller's schematic view was not given since it is complicated as well. In this part, Accelerator Controller's purpose is explained. I/Os of the block is depicted in Figure 3.26.

- It has an internal state machine that shown in Figure 3.31. States numbers are indicated to refer in Figure 3.34. It generates necessary addresses and all read and write signals for the RFs. When "read_write _operation" signal (shown in Figure as "r/w op"), activations are written to memories, weights are mapped to MAC array or memory contents are read. All three operations are performed row by row.

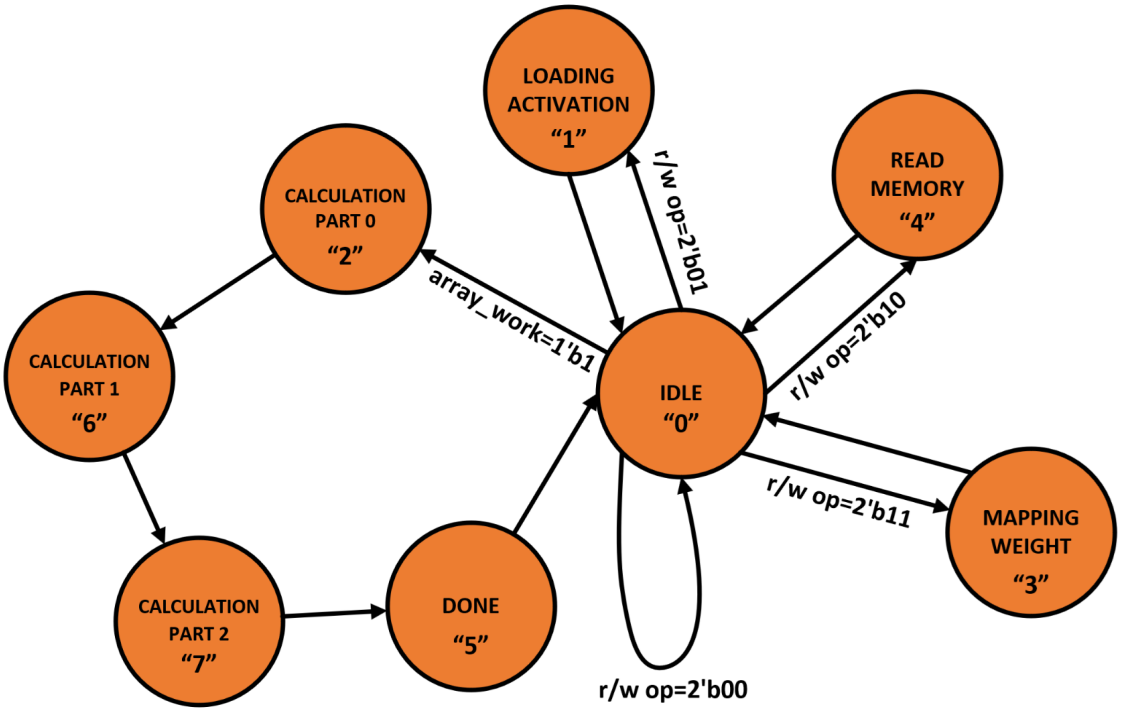

Figure 3.31 Controller State Diagram

- After loading activations and mapping weights, acceleerator is ready to be started, so when array work signal is logic "1", then accelerator starts to calculation.

- In "CALCULATION_PART _0" state, RFs are read and in this state, no result has been come out yet. The Figures 3.6 to 3.9 can be referred to this state.
- In "CALCULATION_PART _1" state, the reading of the first RF has been completed. The first result of the first column comes out and it must be written back to RF. In this state, reading or writing occurs at the same time for different RF blocks. The Figures 3.10 to 3.12 can be referred to this state.

- In "CALCULATION_PART _2" state, the reading of all RFs is completed. In this state, only writing back to memory is performed. The Figures 3.13 to 3.15 can be referred to this state.

- "DONE" state shows that calculation is performed. Results are ready to be read from RFs.

An example of simulation waveform is shown in Figure 3.34. Initially values in RFs are not known. Then activations that are given in Table 3.32, are loaded to RFs row by row, by changing "read_write _operation" signal to 2'b01. Then weights are mapped to MAC Units row by row that are given in Table 3.33, by changing "read_write _operation" signal to 2'b11.

Column1	Column2	Column3	Column4	Column5	Column6	Column7	Column8	Column9	Column10	Column11	Column12	Column13	Column14	Column15	Column16
126	2	64	33	77	47	111	117	107	59	12	73	25	102	89	91
92	44	31	122	64	55	101	14	55	37	93	36	127	52	71	125
80	72	107	103	48	117	39	120	89	122	77	2	11	25	28	69
1	85	125	101	58	106	33	76	24	114	8	70	90	47	116	21
52	124	90	8	125	116	101	9	89	15	35	78	9	13	84	22
118	78	52	124	60	72	53	89	72	88	29	108	91	62	43	118
14	116	66	120	88	100	91	63	80	17	65	98	53	59	53	35
95	95	120	39	40	120	85	60	105	90	75	58	99	68	25	96
19	123	41	38	60	116	22	43	125	20	58	119	21	28	85	74
50	63	73	52	39	5	115	87	86	85	120	36	0	111	22	92
17	29	66	82	95	57	112	87	126	54	16	93	46	84	96	36
113	53	124	81	1	27	106	124	30	93	5	36	7	59	12	8
68	68	79	61	40	105	24	16	103	121	93	109	66	52	93	22
35	60	4	114	57	79	105	124	60	32	92	100	110	112	29	107
107	89	71	86	104	29	87	53	26	38	59	121	65	53	53	98
21	38	6	83	104	99	33	64	27	80	60	125	110	45	24	74

Figure 3.32 An Example Set of Activations

Column1	Column2	Column3	Column4	Column5	Column6	Column7	Column8	Column9	Column10	Column11	Column12	Column13	Column14	Column15	Column16
21	91	-121	-64	-52	-76	72	85	-60	-28	-81	124	119	-114	-98	34
113	38	-111	96	-81	-69	35	-10	75	19	-44	-11	91	-3	9	-69
22	-29	84	36	56	-97	-10	-1	-88	-113	-94	-65	88	-20	-124	115
109	87	-126	-7	-95	-75	-62	110	-27	61	108	-24	74	-49	-79	104
33	-120	12	10	97	-109	-57	127	74	66	-62	-67	72	93	112	-11
26	-70	34	-58	51	13	-120	-80	97	2	116	-53	-84	-119	-30	-71
-127	113	54	-39	16	57	-60	-67	-14	35	-107	-33	15	-119	104	-113
-65	116	-8	48	47	107	125	-92	7	-119	117	5	73	-45	-92	-106
124	-56	90	-7	11	57	-12	-94	-24	90	-91	-117	116	26	119	41
-93	-80	-116	-96	91	118	-86	51	30	-5	-9	-2	-23	83	0	-25
92	-78	0	-46	26	58	96	20	115	16	122	42	65	-75	-94	-84
15	-108	103	124	-127	-41	22	67	-16	127	-127	30	84	-76	-3	-108
103	89	-79	-52	7	-28	14	-4	-36	109	-102	37	60	4	-13	-2
75	72	16	45	103	54	-55	-67	-61	62	-102	-51	81	-100	-92	-123
-26	-27	-12	79	-103	-32	8	67	108	-104	-113	-30	46	-81	113	8
-97	-11	15	28	-42	57	-52	-118	-32	96	-52	115	-86	-76	-4	-93

Figure 3.33 An Example Set of Weights

After that, the accelerator is ready for matrix multiplication. With the "array_work" signal, accelerator starts to operate. After performing the calculations, results are ready to be read. Then, they are read by changing "read_write _operation" signal to 2'b10.

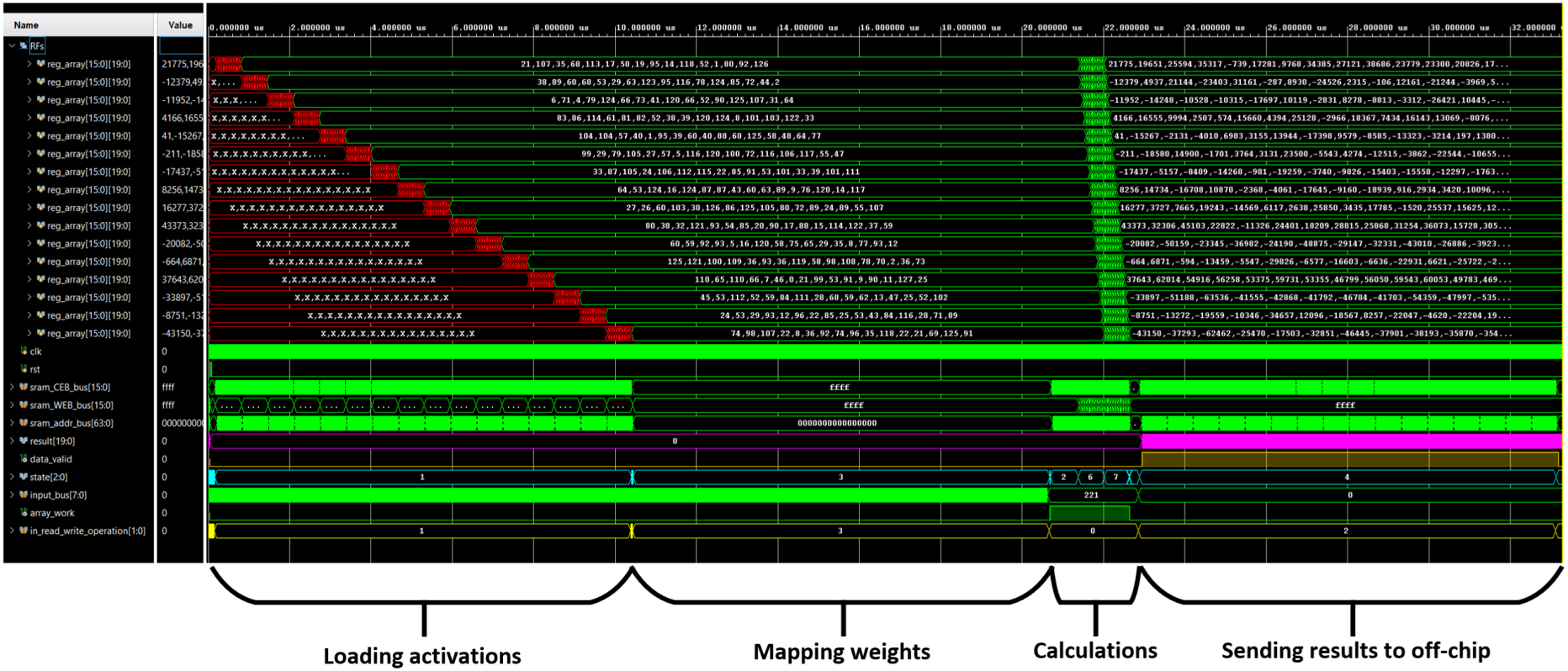

Figure 3.34 An Example Waveform of Accelerator Working

As it can be seen from the waveform 3.1 of the example run, bottleneck of the accelerator is reading activations, mapping weights and sending the result back to off-chip which requires O(N^2) time complexity. Calculating the outputs requires approximately 3xN cycles which has O(N) time complexity as discussed at the end of the section 3.2.

Results and detailed waveform are given in Figure 3.35 and 3.36 respectively. In the both waveform and the result table, $Result_{00}$ is indicated with the purple dot, red and blue arrow indicates the direction of the row and column that corresponds to results in the waveform.

Accelerator has been verified by post-layout simulation as well as the behavioral simulation. Post-layout netlist and Standard Delay Format SDF files have been retrieved from Innovus, and QuestaSim has been employed to perform verification at 200Mhz clock frequency.

Column1	Column2	Column3	Column4	Column5	Column6	Column7	Column8	Column9	Column10	Column11	Column12	Column13	Column14	Column15	Column16
-998	12775	3589	8488	6155	10578	-8488	-11553	-5419	13756	-56074	-6503	59049	-57167	-3509	-40518
26174	15839	-24616	-6125	-11415	-5667	-15404	3446	6346	40910	-37228	8284	48179	-55613	-8585	-29238
17661	549	-21668	-8076	13801	10132	-12001	-8069	12840	-846	1400	-11539	46813	-39478	-29923	-20568
20826	-3969	-17597	13069	197	-10655	-22244	10096	15625	3058	-29900	-22305	46942	-33551	-13632	-17985
23300	-21244	10445	16143	-3214	-22544	-17635	3420	25537	15728	-48062	-25722	49783	-37829	19582	-28665
23779	12161	-26421	7434	-13323	-3862	-12297	2934	-1520	36073	-39235	6621	60053	-53596	-22204	-35475
38686	-106	-3312	18367	-8585	-12515	-15558	916	17785	31254	-26886	-22931	59543	-47997	-4620	-35870
27121	2315	-8813	-2966	9579	4274	-15403	-18939	3435	25868	-43010	-6636	56050	-54359	-22047	-38193
34385	-24526	8278	25128	-17398	-5543	-9826	-9160	25850	28815	-32331	-16603	46799	-41703	8257	-37901
9768	8930	-2831	4394	13944	23500	-3740	-17645	2638	18209	-29147	-6577	53355	-46784	-18567	-46445
17281	-287	10119	15660	3155	3131	-19259	-4061	6117	24401	-48875	-29826	59731	-41792	12096	-32851
-739	31161	-17697	574	6983	3764	-981	-2368	-14569	-11326	-24190	-5547	53375	-42868	-34657	-17503
35317	-23403	-10315	2507	-4010	-1701	-14268	10870	19243	22822	-36982	-13459	56258	-41555	-10346	-25470
25594	21144	-10528	9994	-2131	14900	-8409	-16708	7665	45103	-23345	-594	54916	-38897	-19559	-62462
19651	4937	-14248	16555	-15267	-18580	-5157	14734	3727	32306	-41555	-6636	62014	-51188	-13272	-37293
21775	-12379	-11952	4166	41	-211	-17437	8256	16277	43373	-20082	-664	37643	-33897	-8751	-43150

Figure 3.35 Results Calculated by Software

Figure 3.36 Results Calculated by Accelerator

3.5 Measurement Setup

iW-RainboW-G35M ZU19 Zynq UltraScale+ MPSoC SOM Development Board is going to be employed for providing clock, controlling the chip, sending and receiving the data. Keysight DC Power Supply E3631A is going to be used for both constant and variable power supplies.

Initially activations, weights and results are going to be loaded to FPGA. Necessary control signals are created by the FPGA, to transfer activations and weights, to start the acceleration and to retrieve the result from the accelerator. Received results and loaded results will be compared. Accelerator will be verified functionally at 200Mhz with nominal supply voltage which is $1.2V$.

After verification, bit error rate is going to be observed at different voltage and temperature combinations. Model error probability estimation and measured error probability are going to be compared. Static and dynamic power consumption will be measured by using Logic Analyzer or Oscilloscope.

4. Conclusion

4.1 Summary of Work

With the progression of AI models, AI is being applied in various fields, enhancing its significance. IoT applications have also gained popularity, and there's a growing interest in integrating AI with IoT devices. These devices must be energy-efficient and capable of edge computing, as they often lack a continuous power supply and perform numerous operations. The need for energy efficiency and edge computing stems from not only energy demands but also from requirements for communication bandwidth, low latency, and security. Therefore, it's crucial for DNNs, a prominent AI branch, to be energy-efficient for use in IoT devices.

One method to reduce power consumption is voltage underscaling, as power consumption is proportional to the square of the operating voltage. However, maintaining the same clock frequency while lowering the supply voltage can cause timing errors, which may or may not be tolerable by the DNN model. Traditional transistor-level simulations to study the effects of voltage reduction on the circuit and the DNN model (in terms of inference accuracy) are time-consuming. To speed up the development process of hardware at different voltage levels, beyond those for which gates are characterized, PVT aware fast techniques are necessary instead of extensive simulations.

In this thesis, the probabilistic timing error model which only takes into account voltage supply variation (Rathore et al. (2020)) is verified for 65nm CMOS technology and improved by adding temperature and process variation into the model. The model calculates error probability with acceptable accuracy compared with the MC simulation. However, analysis time reduced about 808 times with respect to 1500 sample MC simulation. Based on the analysis, nominal supply voltage can be

reduced to save power with the trade-off accuracy. To verify the error probability model on the chip, 16x16 systolic MAC array accelerator was designed at 65nm CMOS technology and verified by using various EDA tools.

4.2 Future Work

Probabilistic timing error model calculates the error probability with a reasonable accuracy with a much faster with respect to MC simulations, however model could be improved by improving the gate characterization methods such as, considering different Fan-in and Fan-out capacitance, different slew rates, or increasing the resolution of the temperature and voltage during the extraction of the gate delay. For the small netlist, model calculates lower error probability with respect to MC simulation where, model calculates higher error probability for the large netlist. So, depending on the netlist size, optimization can be applied to model error probability calculation. That brings overhead for the model because the model gets complicated and probably evaluation time increases. Another thing is gate delay definition might be changed based on the voltage level since at lower voltage levels, %50 to %50 delay definition becomes problematic since V_{th} gains more significance at lower operating voltages.

Correlation between voltage and temperature could be found as a value to substitute into model instead of assuming they are totally independent. That makes the model more accurate with respect to real life. To be able to do that, advanced power analysis tools could be employed such as Cadence Voltus.

Lastly, accelerator must be verified functionally by measurement. Then proper activations and weights could be chosen to make sure that switching activity of the gates are high, so that noise on the power lines occur or purposely noise could be added to power supply pins. Then the model must be verified by measurements.

BIBLIOGRAPHY

Amanatidis, P., Iosifidis, G., & Karampatzakis, D. (2022). Comparative evaluation of machine learning inference machines on edge-class devices. In *Proceedings of the 25th Pan-Hellenic Conference on Informatics*, PCI '21, (pp. 102–106)., New York, NY, USA. Association for Computing Machinery.

Atzori, L., Iera, A., & Morabito, G. (2010). The internet of things: A survey.

Chen, J. & Ran, X. (2019). Deep learning with edge computing: A review. *Proceedings of the IEEE*, *PP*, 1–20.

Chen, Y.-H., Krishna, T., Emer, J. S., & Sze, V. (2017). Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, *52*(1), 127–138.

Courbariaux, M., Hubara, I., et al. (2016). Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1.

Dhilleswararao, P., Boppu, S., et al. (2022). Efficient hardware architectures for accelerating deep neural networks: Survey. *IEEE Access*, *10*, 131788–131828.

Dtv, D. R. & Ramana, K. (2019). Accelerating training of deep neural networks on gpu using cuda. *International Journal of Intelligent Systems and Applications*, *11*, 18–26.

Enami, T., Ninomiya, S., & Hashimoto, M. (2009). Statistical timing analysis considering spatially and temporally correlated dynamic power supply noise. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *28*(4), 541–553.

Fauzi, W., Nugroho, S., et al. (2021). Multiple face tracking using kalman and hungarian algorithm to reduce face recognition computational cost. *JAREE (Journal on Advanced Research in Electrical Engineering)*, *5*.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Greengard, S. (2021). *The Internet of Things*. The MIT Press.

Gubbi, J., Buyya, R., et al. (2012). Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, *29*.

Guo, K., Zeng, S., et al. (2018). A survey of fpga-based neural network accelerator.

Halawani, Y., Mohammad, B., et al. (2019). Reram-based in-memory computing for search engine and neural network applications. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, *9*(2), 388–397.

Han, S., Mao, H., & Dally, W. J. (2016). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding.

He, K., Zhang, X., et al. (2015). Deep residual learning for image recognition.

Hochreiter, S. & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*, 1735–80.

Iranfar, A., Terraneo, F., et al. (2017). Thermal characterization of next-generation workloads on heterogeneous mpsocs. In *2017 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, (pp. 286–291).

Jacob, B., Kligys, S., Chen, B., et al. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *2018 IEEE/CVF*

Conference on Computer Vision and Pattern Recognition (CVPR), (pp. 2704–2713)., Los Alamitos, CA, USA. IEEE Computer Society.

Jiao, X., Luo, M., Lin, J.-H., & Gupta, R. (2017). An assessment of vulnerability of hardware neural networks to dynamic voltage and temperature variations. (pp. 945–950).

Jouppi, N., Borchers, A., et al. (2017). In-datacenter performance analysis of a tensor processing unit.

Jouppi, N. P., Young, C., et al. (2017). In-datacenter performance analysis of a tensor processing unit. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, (pp. 1–12).

Judd, P., Delmas, A., Sharify, S., & Moshovos, A. (2017). Cnvlutin2: Ineffectual-activation-and-weight-free deep neural network computing.

Kang, S. & Leblebici, Y. (2003). *CMOS Digital Integrated Circuits Analysis & Design*. McGraw-Hill Series in Electrical and Computer Engineering Series. McGraw-Hill Education.

Kittel, C. (2007). *Introduction to Solid State Physics, 7th Ed.* Wiley India Pvt. Limited.

Kodali, S., Hansen, P., et al. (2017). Applications of deep neural networks for ultra low power iot. In *2017 IEEE International Conference on Computer Design (ICCD)*, (pp. 589–592).

Krizhevsky, A., Sutskever, I., & Hinton, G. (2012a). Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, *25*.

Krizhevsky, A., Sutskever, I., & Hinton, G. (2012b). Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, *25*.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444.

López, P., Montresor, A., et al. (2015). Edge-centric computing. *ACM SIGCOMM Computer Communication Review*, *45*, 37–42.

Mehra, A. (2023). Unveiling the contrasts: Developing small-scale and large-scale language models. Labellerr Blog.

Michalski, R., Carbonell, J., & Mitchell, T. (2013). *Machine Learning: An Artificial Intelligence Approach*. Symbolic Computation. Springer Berlin Heidelberg.

Moon, S., Kim, J., et al. (2015). An accurate on-chip design estimation for mitigating emi effects in a large-scale integration chip. In *2015 IEEE 65th Electronic Components and Technology Conference (ECTC)*, (pp. 757–761).

Muha, B. (2019). What is the iot? introduction to the internet of things. Medium.

Murphy, K. (2012). *Machine Learning: A Probabilistic Perspective*. Adaptive Computation and Machine Learning series. MIT Press.

Nickolls, J. & Dally, W. J. (2010). The gpu computing era. *IEEE Micro*, *30*(2), 56–69.

Overflow, S. (2018). Why can gpu do matrix multiplication faster than cpu? `https://stackoverflow.com/questions/51344018/why-can-gpu-do-matrix-multiplication-faster-than-cpu`.

Owens, J. D. et al. (2008). Gpu computing. *Proceedings of the IEEE*, *96*(5), 879–899.

Pierret, R. (1996). *Semiconductor Device Fundamentals*. Addison-Wesley.

Rathore, M., Milder, P., & Salman, E. (2020). Error probability models for voltage-scaled multiply-accumulate units. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *28*(7), 1665–1675.

Rose, L. K. & Eldridge, S. (2015). The internet of things: An overview understanding the issues and challenges of a more connected world, no.

Ross, S. (2010). *A First Course in Probability*. Pearson Prentice Hall.

Russell, S., Norvig, P., & Davis, E. (2010). *Artificial Intelligence: A Modern Approach*. Prentice Hall series in artificial intelligence. Prentice Hall.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, *3*(3), 210–229.

Shahid, A. & Mushtaq, M. (2020). A survey comparing specialized hardware and evolution in tpus for neural networks. In *2020 IEEE 23rd International Multitopic Conference (INMIC)*, (pp. 1–6).

Sharma, H., Park, J., et al. (2016). From high-level deep neural models to fpgas. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, (pp. 1–12).

Shawahna, A., Sait, S., & El-Maleh, A. (2018). Fpga-based accelerators of deep learning networks for learning and classification: A review. *IEEE Access*, *PP*, 1–1.

Silver, D., Huang, A., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, *529*, 484–489.

Själander, M. & Larsson-Edefors, P. (2009). Multiplication acceleration through twin precision. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, *17*, 1233 – 1246.

Srivatsa, M. (2018). Ai @ edge.

Stilgoe, J. (2017). Machine learning, social learning and the governance of self-driving cars. *Social Studies of Science*, *48*, 030631271774168.

Suda, N., Chandra, V., et al. (2016). Throughput-optimized opencl-based fpga accelerator for large-scale convolutional neural networks. (pp. 16–25).

Sutton, R. & Barto, A. (2018). *Reinforcement Learning, second edition: An Introduction*. Adaptive Computation and Machine Learning series. MIT Press.

Topol, E. J. (2019). High-performance medicine: the convergence of human and artificial intelligence. *Nature Medicine*, *25*(1), 44–56.

Trivedi, D., Bhagchandani, A., et al. (2018). Machine learning in finance. In *2018 IEEE Punecon*, (pp. 1–4).

Tsai, T.-H., Ho, Y.-C., & Sheu, M.-H. (2019). Implementation of fpga-based accelerator for deep neural networks. In *2019 IEEE 22nd International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, (pp. 1–4).

Tsmots, I., Skorokhoda, O., & Rabyk, V. (2019). Hardware implementation of sigmoid activation functions using fpga. In *2019 IEEE 15th International Conference on the Experience of Designing and Application of CAD Systems (CADSM)*, (pp. 34–38).

Turing, A. M. (1950). I.—COMPUTING MACHINERY AND INTELLIGENCE. *Mind*, *LIX*(236), 433–460.

Vanhoucke, V., Senior, A., & Mao, M. (2011). Improving the speed of neural networks on cpus.

Verhelst, M. & Moons, B. (2017). Embedded deep neural network processing: Algo-

rithmic and processor techniques bring deep learning to iot and edge devices. *IEEE Solid-State Circuits Magazine*, *9*(4), 55–65.

Wang, C., Yu, Q., et al. (2016). Dlau: A scalable deep learning accelerator unit on fpga.

Wang, H. & Salman, E. (2017). Closed-form expressions for i/o simultaneous switching noise revisited. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *25*(2), 769–773.

Wang, X., Han, Y., et al. (2020). Convergence of edge computing and deep learning: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, *22*(2), 869–904.

Yan, H., Cherian, H. R., et al. (2020). icelia: A full-stack framework for stt-mram-based deep learning acceleration. *IEEE Transactions on Parallel and Distributed Systems*, *31*(2), 408–422.

Zhang, J., Rangineni, K., Ghodsi, Z., & Garg, S. (2018). Thundervolt: Enabling aggressive voltage underscaling and timing error resilience for energy efficient deep neural network accelerators.

Zhang, J., Sadiqbatcha, S., & Tan, S. X.-D. (2023). Hot-trim: Thermal and reliability management for commercial multicore processors considering workload dependent hot spots. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *42*(7), 2290–2302.

Zhang, S., Gunupudi, P., & Zhang, Q.-J. (2015). Parallel back-propagation neural network training technique using cuda on multiple gpus. In *2015 IEEE MTT-S International Conference on Numerical Electromagnetic and Multiphysics Modeling and Optimization (NEMO)*, (pp. 1–3).

Zhang, Z. & Kouzani, A. (2020). Implementation of dnns on iot devices. *Neural Computing and Applications*, *32*.

Zhao, W. X. et al. (2023). A survey of large language models.