**Design of an Area and Power Optimized Approximate Floating-Point**

**Multiplier using Brent-Kung Adder**

*Submitted in partial fulfilment for the award of the degree of*

**BACHELOR OF TECHNOLOGY**  **In**  **ELECTRONICS & COMMUNICATION ENGINEERING**

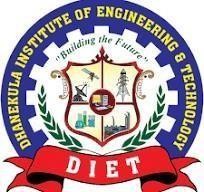
***Submitted by***

**D. HARIKA (218T1A0460) MD. ARSALAN (228T5A0418)**

**G. DURGA PRASAD (218T1A0464) P. ROHINI (218T1A0488)**

***Under the esteemed guidance of***

**MR. S. KRISHNA (Assistant Professor)**



### DHANEKULA INSTITUTE OF ENGINEERING AND TECHNOLOGY (AUTONOMOUS)

### Ganguru, Vijayawada - 521 139

**Affiliated to JNTUK, Kakinada & Approved By AICTE, New Delhi Certified by ISO**

**9001-2015, Accredited by NAAC& NBA**

**APRIL 2025**

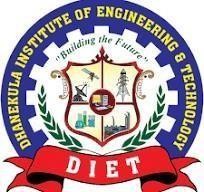
**DHANEKULA INSTITUTE OF ENGINEERING AND TECHNOLOGY**

### Ganguru, Vijayawada - 521 139

**Affiliated to JNTUK, Kakinada & Approved By AICTE, New Delhi Certified by**

**ISO 9001-2015, Accredited by NAAC& NBA**

**APRIL 2025**



**CERTIFICATE**

This is to certify that the project titled“**Design of an Area and Power Optimized Approximate**

**Floating-Point Multiplier using Brent-Kung Adder**” is the bonafide work carried out by **D.**

**HARIKA (218T1A0460), MD. ARSALAN (228T5A0418),G. DURGA PRASAD**

**(218T1A0464)**,  **P. ROHINI (218T1A0488)** **student of B Tech(ECE)** of Dhanekula Institute of Engineering & Technology, affiliated to JNT University, Kakinada, AP(India) during the academic year 2024-2025, in partial fulfilment of the requirements for the award of the degree of Bachelor of Technology(Computer Science and Engineering) and that the project has not formed the basis for the award previously of any other degree, diploma, fellowship or any other similar title.

|  |  |
| --- | --- |
| **Signature of the GUIDE** | **Signature of the HOD** |
| **S. KRISHNA** | **Dr. VAMSHI KRISHNA** |
| **(Assistant Professor)** | **(Professor & HOD)** |

**Signature of the External Examiner**

**ACKNOWLEDGMENT**

First, we sincerely salute our esteemed institution **DHANEKULA INSTITUTE OF**

**ENGINEERING AND TECHNOLOGY** for giving us this opportunity for fulfilling our project.

We express our sincere thanks to our beloved Principal **Dr. K. RAVI** for providing all the lab facilities and library required for completing this project successfully.

This project work would not have been possible without the support of many people. We wish to express our gratitude to **Dr. VAMSI KRISHNA**, Ph.D., H. O. D. ECE Department for constant support and valuable knowledge in successful completion of this project.

We are glad to express our deep sense of gratitude to **Mr. S. KRISHNA,** Assistant Professor, for abundant help and offered invaluable assistance, support and guidance throughout this work.

We thank all who have rendered help directly or indirectly in the completion of this project successfully.

**PROJECT ASSOCIATES**

**D. HARIKA (218T1A0460)**

**MD. ARSALAN (228T5A0418)**

**G. DURGA PRASAD (218T1A0464)**

**P. ROHINI (218T1A0488)**

**DECLARATION**

We hereby declare that the project entitled “**Design of an Area and Power Optimized Approximate Floating-Point Multiplier using Brent-Kung Adder**” submitted for the B.Tech. (ECE) degree is our original work and the project has not formed the basis for the award of any other degree, diploma, fellowship or any other similarities.

**Signature of the students:**

|  |  |
| --- | --- |
| **D. HARIKA** | **(218T1A0460)** |
| **MD.ARSALAN** | **(228T5A0418)** |
| **G. DURGA PRASAD** | **(218T1A0464)** |
| **P. ROHINI** | **(218T1A0488)** |

**Place: Vijayawada**

**Date:**

**Contents**

**VISION – MISSION - PEOs ...............................................................................I-II**

**STATEMENT OF PO`s & PSO`s ....................................................................... III**

**PROJECT MAPPING - PO`s & PSO`s ................................................................ V**

**List of Figures: ..................................................................................................... VII**

**List of Tables ...................................................................................................... VIII**

**Abstract .................................................................................................................IX**

**CHAPTER-1 ........................................................................................................... 1**

**Introduction ..............................................................................................................2**

**1.1Introduction** ......................................................................................................2

**1.1 Background** ..................................................................................................... 3

**1.2 Motivation for Approximate Computing** ..................................................... 3

**1.3 Problem Statement** ......................................................................................... 3

**1.4 Objectives of the Project** ................................................................................ 3

**1.5 Scope of the Project** .................................................................................... 3-5

**CHAPTER-2 ...........................................................................................................6**

**Literature Survey ................................................................................................. 7-9**

**CHAPTER-3 ...........................................................................................................10**

**Existing System ......................................................................................................11**

**3.1 Approximate Floating -point Using CLA Adder** ................................. 11-15

**CHAPTER-4 ...........................................................................................................16**

**Proposed Method ...................................................................................................17**

4.1Brent Kung Parallel Prefix Adder ....................................................................17

4.2Key Concepts of the Brent-Kung Adder ................................................... 18-19

4.3 Block Diagram .......................................................................................... 21-24

4.4Advantages .......................................................................................................25

**CHAPTER-5 ...........................................................................................................26**

**SOFTWARE TOOL ........................................................................................ 27-44**

**CHAPTER-6 ...........................................................................................................45**

**Results ............................................................................................................... 46-51**

**CHAPTER-7 ....................................................................................................... 52**

**Conclusion and Future Scope ........................................................... 53**

**FutureScope........................................................................................ 54**

**References ............................................................................................................55**

**Appendex .......................................................................................................... 57-66**

**DHANEKULA INSTITUTE OF ENGINEERING & TECHNOLOGY**

**Department of Electronics & Communication Engineering**

# VISION – MISSION - PEOs

Vision/Mission/PEOs

|  |  |
| --- | --- |
| Institute Vision | Pioneering Professional Education through Quality |
| Institute Mission | Providing Quality Education through state-of-art infrastructure, laboratories and committed staff.    Molding Students as proficient, competent, and socially responsible engineering personnel with ingenious intellect.    Involving faculty members and students in research and development works for betterment of society. |
| Department Vision | Pioneering Electronics & Communication Engineering education and research to elevate rural community |
| Department Mission | Imparting professional education endowed with ethics and human values to transform students to be competent and committed electronics engineers.  Adopting best pedagogical methods to maximize knowledge transfer.  Having adequate mechanisms to enhance understanding of theoretical concepts through practice.  Establishing an environment conducive for lifelong learning and entrepreneurship development.  To train as effective innovators and deploy new technologies for service of society. |

|  |  |
| --- | --- |
| Program  Educational  Objectives (PEOs) | PEO1: Shall have professional competency in electronics and communications with strong foundation in science, mathematics and basic engineering.    PEO2: Shall design, analyze and synthesize electronic circuits and simulate using modern tools.    PEO3: Shall Discover practical applications and design innovative circuits for Lifelong learning.    PEO4: Shall have effective communication skills and practice the ethics consistent with a sense of social responsibility. |

# STATEMENT OF PO`s & PSO`s

|  |  |
| --- | --- |
|  | **Program Outcomes** |
| PO1 | **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals and engineering programs. |
| PO2 | **Problem analysis**: Identify, formulate, review research literature, and analyse complex Engineering problems reaching substantiated conclusions using first principles of Mathematics, natural sciences, and engineering sciences. |
| PO3 | **Design/development of solutions**: design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental Considerations. |
| PO4 | **Conduct investigations of complex problems**: Use research-based knowledge and research Methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions. |
| PO5 | **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations. |
| PO6 | **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice. |
| PO7 | **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development. |
| PO8 | **Ethics:** Apply ethical principles and commit to professional ethics and responsibility and norms of the engineering practice. |
| PO9 | **Individual and team work:** Function effectively as an individual and as a member or leader in diverse teams and in multidisciplinary settings. |
| PO10 **Communication:** Communicate effectively on complex engineering activities with the Engineering community and with society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions. | |

PO11 **Project management and finance:** Demonstrate knowledge and understand of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12 **Life-long learning**: Recognize life-long the need for and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

### Program Specific Outcomes

PSO1 Have expertise in linear & digital circuits, signal processing, communications, VLSI and embedded systems.

PSO2 Design and Development of Innovative products relevant for the society.

# PROJECT MAPPING - PO`s & PSO`s

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Project Title** | **PO**  **1** | **PO**  **2** | **PO**  **3** | **PO**  **4** | **PO**  **5** | **PO**  **6** | **PO**  **7** | **PO**  **8** | **PO**  **9** | **PO1**  **0** | **PO1**  **1** | **PO1**  **2** |
| **Design of an Area and Power**  **Optimized**  **Approximate**  **Floating-Point**  **Multiplier Using**  **Brent-**  **Kung Adder** | 2 | 3 | 3 | 3 | 3 | 3 | 3 | - | 3 | 3 | 3 | 3 |

3-High 2-Medium 1- Low

**Justification of Mapping of Project with Program Outcomes:**

1. The knowledge of mathematics, science, engineering fundamentals and engineering programs are strongly correlated to all course outcomes.
2. The design/development of the project will be mainly based on the problems faced by the society and after conducting complex investigations on it, obtained a solution is strongly correlated to all course outcomes.
3. Application of Ethical principles is not correlated to all course outcomes**.**

### Project vs PSOs Mapping

|  |  |  |
| --- | --- | --- |
| Project Title | PSO1 | PSO2 |
| **Design of an Area and Power Optimized Approximate**  **Floating-Point Multiplier Using Brent-Kung Adder** | 3 | 3 |
|  |  |  |

3-High 2-Medium 1- Low

**Justification of Mapping of Project with Program Specific Outcomes:**

1. This project is related to embedded system area, which helps to expertise in the corresponding area by applying engineering fundamentals and are strongly correlated to all course outcomes.
2. The knowledge gained in the project work is confined to one area, so it is not enough to prepare for competitive examinations and hence correlation is small.

# 

# List of Figures:

**Title of the Figure Pg-No**

1. Architecture of floating point using CLA Adder 14
2. Error correction model using CLA Adder 15
3. General Multiplication 15
4. Architecture of the proposed approximate multiplier

With Brunt-Kung Adder 21

1. Error Correction Model with Brent-Kung Adder 22
2. General Multiplication of Brunt-Kung Adder: 24

# List of Tables

1.comparison Between Carry Look-ahead Adder and Brent-Kung Adder pg-no-51

List of Abbreviations

Abbreviation Description

LUT Look Up Tables

FF Flip-Flops

IOB Input and Output Blocks

BUFG Buffers

# Abstract

Traditional hardware implementations of floating-point multipliers prioritize accuracy but consume significant power and area. However, proposed technology is better than existing system and it can tolerate slight precision loss, making approximate computing a viable solution for enhancing energy efficiency without compromising overall system performance.

This project introduces an efficient approximate floating-point multiplier that integrates a BrentKung Adder to optimize area and power consumption. The Brent-Kung Adder is chosen due to its minimal logic depth and efficient parallel prefix structure, which significantly reduces propagation delay and hardware overhead compared to traditional adders. The multiplier is designed to support runtime reconfigurable precision and clock frequency, allowing dynamic trade-offs between accuracy and energy savings.

The proposed design is implemented in Verilog HDL and synthesized using Xilinx Vivado. The floating-point multiplier is developed using Verilog HDL, incorporating the Brent-Kung Adder to optimize addition operations. Functional simulations are conducted to verify the correctness of the multiplier under different levels of precision. The design is synthesized on FPGA platforms to evaluate area utilization, power consumption, and timing performance. The proposed multiplier is benchmarked against conventional approximate and exact multipliers to analyze improvements in hardware efficiency.

Future extensions of this work include support for adaptive precision control based on real-time workload variations, further optimizations in multiplier architecture to enhance speed and reduce latency, and improved power , area efficiency. Integrating Brent-Kung Adders with approximate computing techniques provides a practical and efficient solution for next-generation AI hardware accelerators, bridging the gap between accuracy

# 

# CHAPTER-1

# Introduction

## 1.1Introduction

In modern computing applications such as Deep Neural Networks , balancing computational accuracy with energy and area efficiency is critical. This project introduces an Approximate Floating-Point Multiplier designed to optimize area and power consumption, incorporating a Brent-Kung Adder for reduced logic complexity and a Frequency and Precision Controller for dynamic performance adjustment. The controller operates across three clock domains (CLK1, CLK2, CLK3) corresponding to different precision levels, allowing runtime reconfiguration to trade off between power, speed, and accuracy based on the application’s needs. CLK1 enables the fastest operation with minimal precision, while CLK3 provides higher precision at a lower clock rate, with CLK2 offering a balance between the two. This flexibility enables significant energy savings, particularly in applications with variable accuracy requirements.

The Brent-Kung Adder is integrated into the design to minimize the logic size of carry propagation, reducing overall area compared to traditional adder structures such as Carry Lookahead Adders. The reduced complexity of the Brent-Kung architecture leads to substantial gains in both area and power efficiency. Additionally, the design employs an error correction mechanism that adjusts the precision of the floating-point multiplication dynamically. The module corrects for approximation errors at different levels, depending on the precision mode in use. The result is a system that can deliver high accuracy when necessary while allowing for lower precision and reduced power usage in less critical computations.

To evaluate the performance of the proposed multiplier, extensive experiments were conducted on Synopsys Design Compiler and Xilinx FPGA platforms. The design is developed in Verilog HDL and synthesized using Xilinx Vivado, allowing for a direct comparison against traditional exact

## 1.1 Background

In recent years, the rise of computationally intensive applications—particularly those involving artificial intelligence, machine learning, and signal processing—has driven demand for hardware that balances performance with energy and area efficiency. Floating-point operations, especially multiplication, are among the most resource-hungry components in such systems. While exact computation ensures high accuracy, it often comes at the cost of increased power consumption and silicon area.

## 1.2 Motivation for Approximate Computing

Approximate computing provides an opportunity to relax precision constraints in favor of improvements in power, area, and speed. This paradigm is especially relevant in applications where perfect accuracy is not critical—such as image processing, DNN inference, and edge analytics. These scenarios often tolerate small errors while benefiting greatly from reduced hardware complexity and energy use. Hence, the motivation is to design a multiplier that can adaptively trade accuracy for efficiency as needed.

## 1.3 Problem Statement

Traditional floating-point multipliers are optimized for precision, making them unsuitable for energy- or area-constrained environments. Furthermore, they lack the ability to adapt dynamically to workload precision requirements. This project aims to address this limitation by designing an approximate floating-point multiplier that is both **reconfigurable** and **resourceefficient**, without significantly compromising output accuracy.

## 1.4 Objectives of the Project

* To design and implement an approximate floating-point multiplier in Verilog HDL.
* To integrate a Brent-Kung Adder for efficient carry propagation.
* To develop a Frequency and Precision Controller supporting three precision modes.
* To evaluate the design on Synopsys Design Compiler and Xilinx FPGA platforms.
* To compare performance metrics (area, power, speed) against traditional designs.
* To demonstrate the suitability of the proposed multiplier for edge and ML applications.

## 1.5 Scope of the Project

This project focuses on hardware-level design and evaluation of an energy-efficient, dynamically reconfigurable approximate multiplier. It does not cover full system-level integration (e.g., with a CPU or GPU) but aims to provide a modular and scalable unit suitable for inclusion in such systems. Both ASIC and FPGA implementation results are considered to assess practical feas where different layers of a neural network may require varying levels of computational accuracy. In many DNN architectures, initial layers focus on high-level feature extraction, which may tolerate minor precision loss, whereas final layers demand higher accuracy for classification and decision-making. By dynamically adjusting the precision of the multiplication process, this design optimizes power consumption without significantly impacting the overall performance of the neural network.

Furthermore, the proposed multiplier design addresses key challenges associated with approximate computing, particularly the balance between error tolerance and hardware efficiency. Approximate multipliers typically introduce errors that accumulate over successive computations, leading to potential degradation in output quality. However, the integrated error correction module in this design mitigates these issues by applying different levels of correction depending on the operating precision mode. This ensures that computational errors remain within acceptable bounds while leveraging the benefits of approximate arithmetic to achieve energy efficiency.

Another critical aspect of this work is its potential integration into existing FPGA-based accelerators and hardware accelerators for AI applications. Given the increasing adoption of FPGA-based deep learning accelerators, this design can be seamlessly incorporated into existing pipelines to enhance computational efficiency. The FPGA-friendly nature of the design ensures that it can be synthesized and deployed effectively without significant modifications to existing architectures. Additionally, its compatibility with hardware reuse techniques makes it a viable candidate for low-power embedded systems, where computational resources are limited.

The power efficiency of the proposed multiplier stems from its ability to selectively reduce switching activity in low-precision modes. By dynamically adjusting clock frequencies and logic activity, the design minimizes unnecessary power dissipation, leading to overall improvements in energy consumption. This aspect is crucial for battery-powered applications such as mobile devices, IoT sensors, and wearable electronics, where extending battery life is a key consideration. The ability to balance computational performance with energy constraints makes this design particularly well-suited for next-generation computing systems.

From an architectural perspective, the Brent-Kung Adder's role in optimizing logic size and carry propagation delays is pivotal. Unlike traditional adders, which rely on complex carry propagation mechanisms, the Brent-Kung structure divides the carry computation into smaller, more efficient stages. This results in a significant reduction in gate count and propagation delay, contributing to the overall efficiency of the multiplier. Additionally, the modular nature of the Brent-Kung Adder facilitates easier integration into FPGA architectures, further enhancing its practicality for realworld implementations.

The impact of the proposed multiplier on DNN performance is further analyzed by evaluating its accuracy degradation across various network architectures. Experimental results indicate that for many deep learning workloads, the accuracy loss due to approximation is minimal and does not significantly affect the final inference outcomes. This underscores the feasibility of using approximate multipliers in AI applications without compromising the integrity of the computations. By leveraging controlled approximation, the design achieves a favorable trade-off between power savings and computational reliability.

Future research directions for this work include exploring additional error mitigation techniques, such as adaptive precision scaling based on real-time accuracy feedback. By incorporating machine learning-based adaptive control mechanisms, the multiplier can further optimize precision dynamically, enhancing its effectiveness across diverse workloads. Additionally, investigating alternative adder architectures for further power reduction may yield additional improvements in computational efficiency. Expanding the scope of analysis to include ASIC implementations can also provide insights into the feasibility of deploying this design in commercial AI accelerators.

Overall, this project presents a novel approach to designing an energy-efficient approximate floating-point multiplier, combining hardware reuse techniques, dynamic precision control, and low-power design principles. The proposed methodology enables significant reductions in power and area while maintaining sufficient accuracy for AI-driven applications. Experimental results validate the feasibility and advantages of this approach, demonstrating its potential for widespread adoption in energy-constrained computing environments.

# 

# CHAPTER-2

**Literature Survey**

1. Approximate Computing for Deep Neural Networks: This survey underscores the increasing reliance on approximate computing techniques in DNN accelerators to meet energy and area constraints, particularly in edge devices. It thoroughly reviews how reducing arithmetic precision—such as using 8-bit or mixed-precision formats instead of 32-bit floats—leads to a significant drop in energy usage without heavily affecting inference accuracy. Various approximate multiplier designs are examined, highlighting their role in accelerating matrix operations, the backbone of neural network computations. It also covers the design of bit-level truncation, significance-driven computation, and hybrid precision schemes, illustrating real-world implementations that offer energy savings upwards of 50%.
2. Brent-Kung Adder: The Brent-Kung Adder is analyzed as a critical building block in arithmetic units, particularly for low-power approximate designs. The paper presents its logarithmic delay and reduced fan-out properties, making it well-suited for high-speed operations in constrained environments. Compared to other prefix adders like Kogge-Stone or Ladner-Fischer, Brent-Kung shows a superior balance between area and delay, making it a favorable choice in the design of approximate multipliers where silicon footprint is a concern. The adder’s efficiency directly contributes to reduced power dissipation in arithmetic-intensive circuits.
3. Error-Tolerant Floating-Point Arithmetic: This literature survey provides a solid foundation for floating-point approximations, emphasizing the fact that not all bits in a floating-point number contribute equally to output accuracy. The concept of "selective precision" is explored, where less significant mantissa bits are approximated or discarded to reduce switching activity in hardware. Applications in convolutional neural networks (CNNs) and recurrent networks are discussed, where such techniques are used during inference. Additionally, the paper evaluates the use of compensatory error detection and correction mechanisms like residue checking and parity to ensure reliability when required.
4. Reconfigurable Clock Systems for Low-Power VLSI: This paper introduces adaptive clocking as a novel control mechanism to manage power and performance trade-offs dynamically. It explores variable clock frequency and voltage scaling depending on workload precision needs. For multipliers, this translates to running lower-precision modes at lower voltages and higher frequencies, while switching to full-precision at higher power only when needed. The integration of these clock systems with precision-scalable arithmetic units offers promising directions for energy-aware DNN hardware, especially when combined with runtime workload profiling.

1. Hardware-Efficient Approximate Multipliers: The study conducts a side-by-side evaluation of various approximate multiplier designs, such as Broken Array Multipliers, Lower-part OR Adders (LOA), and Error-Tolerant Adders (ETA). It benchmarks these against traditional multipliers on parameters like area, delay, power, and error metrics (e.g., Mean Error Distance and Worst-Case Error). The study reveals that some designs, although highly energy-efficient, introduce unacceptable error for sensitive applications, while others like LOA strike a better compromise. It underscores the importance of choosing a design based on application-level tolerance.
2. Energy-Efficient Multipliers for Deep Learning Inference on Edge Devices. Focusing on resource-constrained environments, this paper highlights the pivotal role of energy-efficient multipliers in enabling DNN inference at the edge. It reviews fixed-point and approximate floating-point multiplier implementations and proposes novel methods like operand-aware truncation and partial product skipping. It also examines workload-aware dynamic precision scaling in mobile inference engines like Edge TPU, showing that hybrid-precision computing can reduce power consumption by 40–60% without significantly affecting top-1 accuracy in classification tasks.
3. Adaptive Precision in Floating-Point Arithmetic for Performance and Power Optimization. This paper introduces control logic mechanisms for adaptive precision computing, where hardware can switch between different levels of accuracy based on runtime input statistics or application phase. For instance, in CNNs, early layers may use lower precision, while later layers use higher precision to refine features. It presents a modular floating-point unit design with configurable mantissa width and supports software-hardware co-design approaches for dynamic bit-width tuning. This flexibility is key for designs targeting a wide range of workloads with varying accuracy demands.

1. Power-Aware Arithmetic Circuits: A Survey of Techniques and Implementations. This extensive survey discusses circuit-level and architecture-level optimizations for reducing power in arithmetic units, such as operand gating, speculative computation, and clock/data gating. It highlights how combining low-power adders (like Brent-Kung or Han-Carlson) with approximate multipliers leads to compounded energy savings. The paper categorizes techniques into designtime and runtime optimizations, promoting a holistic approach to power-aware design in nextgeneration VLSI systems.

1. Approximate Floating-Point Arithmetic for Machine Learning Accelerators. A detailed exploration of how approximation in floating-point arithmetic is exploited in hardware accelerators like GPUs and TPUs. It compares different techniques such as leading-zero approximation, mantissa segmentation, and exponent pruning. The paper also introduces the concept of *approximate fused multiply-add (FMA)* units tailored for matrix-matrix multiplications, where error tolerance is managed at the algorithmic level (e.g., in weight quantization). It emphasizes cooptimization of algorithm and hardware for best efficiency gains.
2. Error Correction in Approximate Computing: Strategies and Hardware Implementations. This paper explores how lightweight error correction techniques can be embedded into approximate computing pipelines to maintain reliability without incurring significant overhead. It presents examples such as dynamic error compensation in approximate multipliers, and dual-path computation where approximation is used first and corrected only if a threshold of error is exceeded. Hybrid techniques that use both approximation and error correction ensure a graceful degradation of performance, which is highly desirable in critical applications like autonomous driving or medical diagnosis using neural networks.

# CHAPTER-3

# 

## Existing System

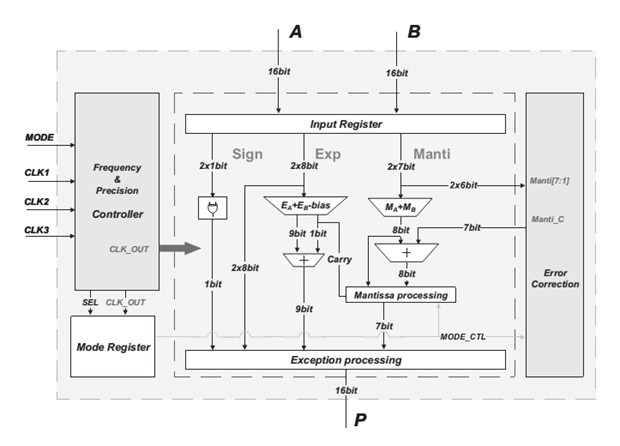
## 3.1 Approximate Floating -point Using CLA Adder

In the past decade,

Approximate computing leverages intrinsic error tolerance of applications to provide resource and energy efficient design at the cost of acceptable accuracy loss. In massive MAC operations of brent-kung adder is better than CLA adder, multiplication consumes most of the total energy and hence low power approximate multipliers that can achieve almost the same accuracy as the precise 32-bit floating-point (FP32) multipliers have been proposed. However, the current approximate fixed-point multipliers require to quantify the parameters accordingly and cannot replace all the FP32 multipliers. Second, they have the same precision on approximate multiplication operations without considering the fact that different layers in brent-kung adder have different tolerances for accuracy drops. Therefore, the existing approximate multipliers still have some limitations to accelerate better methodology without performance degradation.

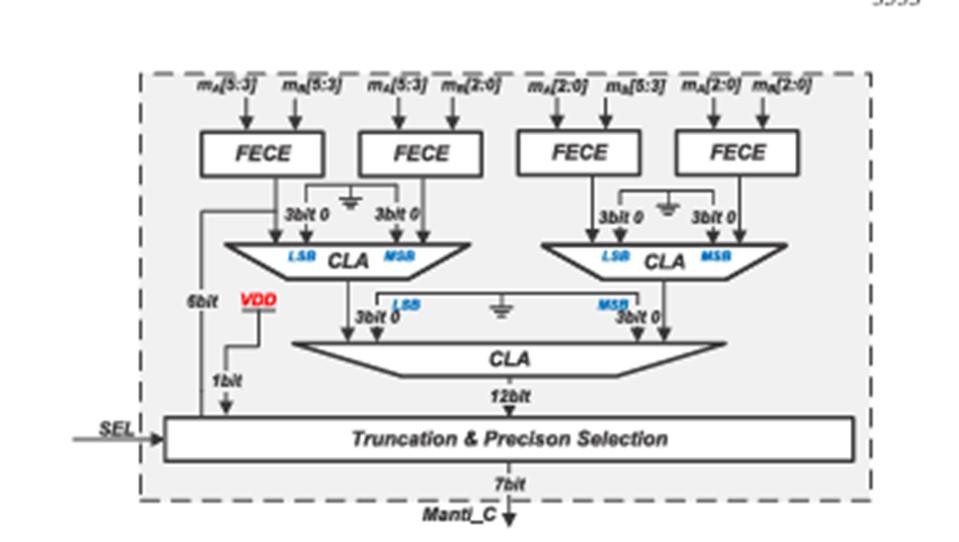
In this brief, we focus on improving hardware efficiency and performance of computationintensive applications without compromising accuracy. An approximate floating-point multiplier is proposed to provide reconfigurability of both multiplication precision and clock frequency at runtime. The novel approximation strategy and a well-designed architecture help the proposed approximate multiplier to achieve hardware and energy efficiency. The reconfiguration of clock signal and the precision modes are determined dynamically according to different task requirements so that the proposed method achieves high efficiency while ensuring the loss of inference accuracy is acceptable and controllable. Our main contributions are as follows:

1. An approximate floating-point multiplier with an error correction option is proposed based on the logarithmic approximation algorithm. The error correction module offers three precision modes and is implemented using a hardware reuse methodology, ensuring improved efficiency while maintaining flexibility in precision control. The proposed approach allows for significant reductions in hardware complexity and power consumption, making it an ideal choice for resource-constrained applications, such as deep learning accelerators and edge computing devices.
2. A novel control module is designed to dynamically reconfigure the clock frequency at runtime based on precision requirements. This reconfigurability allows for a flexible balance between energy efficiency and computational accuracy, ensuring optimized performance under different operational conditions. The design is free from glitches and metastability problems, guaranteeing stable and reliable operation even in high-speed computing environments. The clock control mechanism also helps in reducing dynamic power consumption, further enhancing the energy efficiency of the system.
3. Extensive experiments have been conducted to evaluate the proposed approximate multiplier on industry-standard tools, including the Synopsys Design Compiler and Xilinx FPGA platforms. These evaluations provide a comprehensive analysis of the design's effectiveness in real-world hardware implementations. Compared with traditional precise multipliers, our design demonstrates significant improvements in speed, hardware overhead, and power consumption. These enhancements make it a promising solution for applications where strict precision is not always necessary, such as deep neural networks and artificial intelligence inference tasks.
4. When compared to state-of-the-art approximate multipliers, our design exhibits superior accuracy while maintaining the benefits of reduced hardware complexity and power consumption. The controlled error characteristics ensure that the introduced approximation does not significantly affect the output quality. The design achieves smaller and more predictable errors, which can be carefully tuned to meet specific application requirements. This fine-grained control over accuracy makes it possible to apply the proposed multiplier to a wide range of computational workloads without compromising performance.
5. The accuracy loss in CLA adder inference tasks is found to be negligible, as brent-kung adder is better than CLA adder inherently tolerate minor computational inaccuracies without a significant impact on overall performance. This makes the proposed approximate multiplier particularly well-suited for accelerating better technology, enabling energyefficient AI processing with minimal accuracy degradation. The power and area savings achieved through this design allow for more compact and cost-effective AI hardware implementations, making it highly beneficial for modern AI accelerators and embedded systems.
6. The proposed multiplier is designed for easy integration into existing hardware architectures, supporting seamless implementation in FPGA and ASIC-based accelerators. The efficient utilization of logic resources and the incorporation of approximate computing techniques contribute to the overall scalability of the design. This makes it adaptable for a wide range of applications, including high-performance computing, digital signal processing, and embedded AI systems.
7. By leveraging approximate computing methodologies, this work contributes to the ongoing advancements in energy-efficient hardware design. The ability to trade off precision for efficiency is a key feature that enables substantial power savings without significant loss of computational effectiveness. This approach paves the way for further research into adaptive precision techniques and energy-aware computing paradigms.
8. Future extensions of this work may include further optimizations in multiplier architecture to enhance speed and reduce latency. Additionally, integrating adaptive precision control based on real-time workload variations can enhance the efficiency of the design even further. Implementing the proposed architecture in ASIC technology can also provide improved power and area benefits, making it an attractive solution for next-generation computing platforms



**Figure-1 Architecture of approximate Floating-Point:**

From Figure-1 The multiplication core simplifies floating-point multiplication into addition in the logarithmic domain, enhancing computational efficiency and reducing energy consumption. The frequency and precision controller allows seamless switching of three unrelated clock signals and sends the precision control signal to the multiplication core via the mode register, enabling runtime reconfiguration of precision and frequency



**Figure-2 Error Correction Model**

t employs Fixed-Error Carry Estimator (FECE) blocks to generate partial products from segmented mantissa inputs, significantly reducing complexity. These partial results are then summed using fast Carry Lookahead Adders (CLA) for both LSB and MSB portions, culminating in a 12-bit final CLA stage. A truncation and precision selection unit, controlled by a SEL signal, enables runtime configurability, allowing dynamic adjustment of output precision (7-bit mantissa) based on application needs. This design supports power savings through controlled accuracy, efficient logic depth, and reduced switching activity, making it ideal for low-power, high-performance edge AI applications.

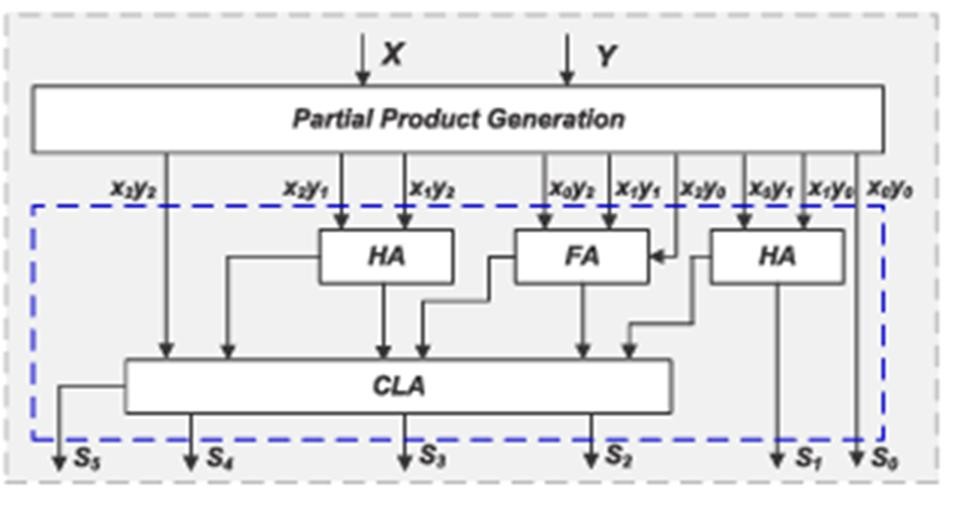


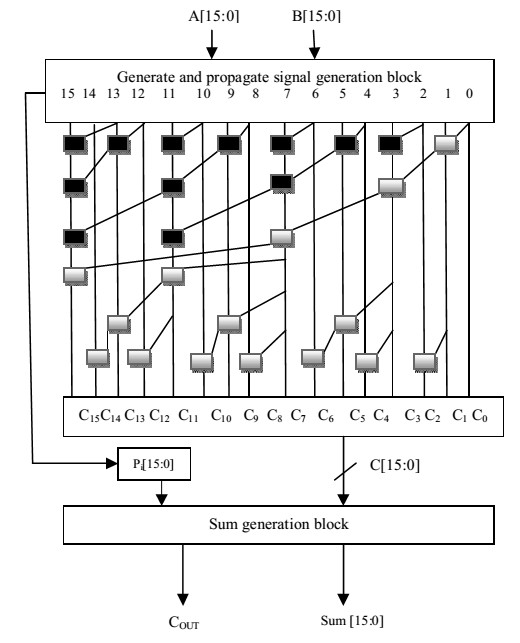
Figure 3 General Multiplication Method

# CHAPTER-4

# 

# Proposed Method

## 4.1Brent Kung Parallel Prefix Adder



The Brent-Kung Parallel Prefix Adder is a highly optimized binary adder architecture designed to improve addition speed while minimizing hardware complexity. As a member of the parallel prefix adder family, it is primarily used in digital circuits to accelerate carry propagation, which is a critical factor in determining the speed of binary addition. The Brent-Kung Adder achieves an efficient trade-off between area and speed, making it a suitable choice for applications where reduced power consumption and minimal logic resources are desired, such as FPGA and ASIC implementations.

The architecture of the Brent-Kung Adder is based on a systematic approach to computing carry bits using a tree-like structure. Unlike the Ripple Carry Adder (RCA), which suffers from linear propagation delay due to sequential carry computation, and the Carry Lookahead Adder (CLA), which requires additional logic complexity, the Brent-Kung Adder strikes a balance between both.

It reduces the number of logic levels needed to compute the carry by employing a hierarchical prefix computation method, leading to a logarithmic time complexity of O(log2N), where N is the number of bits.

The Brent-Kung Adder consists of three main stages: pre-computation, prefix computation, and final sum generation. In the pre-computation stage, the generate and propagate signals for each bit position are determined. The prefix computation stage utilizes a parallel prefix tree to efficiently compute the carry signals, which are then used in the final stage to determine the sum bits. The tree-based structure of the prefix computation stage significantly reduces the number of logic gates required compared to other parallel prefix adders like the Kogge-Stone Adder, leading to lower power consumption and reduced silicon area.

One of the notable advantages of the Brent-Kung Adder is its reduced fan-out, meaning that each logic stage drives fewer subsequent gates, thereby reducing power consumption and improving circuit stability. While it does introduce slightly more delay than the Kogge-Stone Adder due to its more compact prefix tree, the trade-off in terms of area efficiency makes it ideal for applications where both power and space constraints are critical.

Due to its inherent efficiency, the Brent-Kung Adder is widely adopted in FPGA-based designs, digital signal processors (DSPs), and arithmetic units within microprocessors. Its ability to balance speed and resource utilization makes it a valuable component in modern VLSI design, particularly in applications requiring optimized hardware for arithmetic operations.

## 4.2Key Concepts of the Brent-Kung Adder

**1. Prefix Addition Concept:**

* Binary addition is performed by generating carries that propagate through the bits of the operands. Prefix adders use a tree structure to compute these carries in parallel, reducing the overall addition time compared to simple ripple-carry adders.
* The Brent-Kung adder is based on the carry-look ahead principle but uses a parallel prefix tree to compute the generate and propagate signals more efficiently.

**2. Generate and Propagate Signals:**

* For each bit pair, the generate (G) and propagate (P) signals are computed:
* `Generate (G) = A \* B` (i.e., if both input bits A and B are 1, a carry is generated).
* `Propagate (P) = A + B` (i.e., if either A or B is 1, a carry can propagate through this bit).
* These signals are then combined in a hierarchical manner to determine the carry for each bit.

**3. Parallel Prefix Structure:**

* The Brent-Kung adder uses a tree-like structure that allows the computation of carry signals in parallel, which reduces the critical path delay.
* The prefix computation is divided into two stages:

1. Upward Tree (Reduction stage): The generate and propagate signals are computed for each bit, moving from the least significant bit (LSB) to the most significant bit (MSB). In this stage, intermediate carry information is generated in a logarithmic number of steps (O(log n)).
2. Downward Tree (Post-processing stage): The carry information is distributed to the lower bits by tracing back down the tree.

**4. Low Logic Complexity:**

* One of the key features of the Brent-Kung adder is that it minimizes the number of logic gates used. It achieves this by having fewer nodes in the prefix tree than other designs (like KoggeStone), resulting in less hardware overhead.
* Although it may not be as fast as some other parallel prefix adders (like Kogge-Stone or Sklansky), it strikes an excellent balance between speed and area efficiency.

**5. Latency:**

- The depth of the Brent-Kung adder is logarithmic (O(log n)), where \*n\* is the number of bits in the adder. This means that as the bit-width increases, the delay grows logarithmically rather than linearly, making it faster than ripple-carry adders but slower than more complex designs like Kogge-Stone.

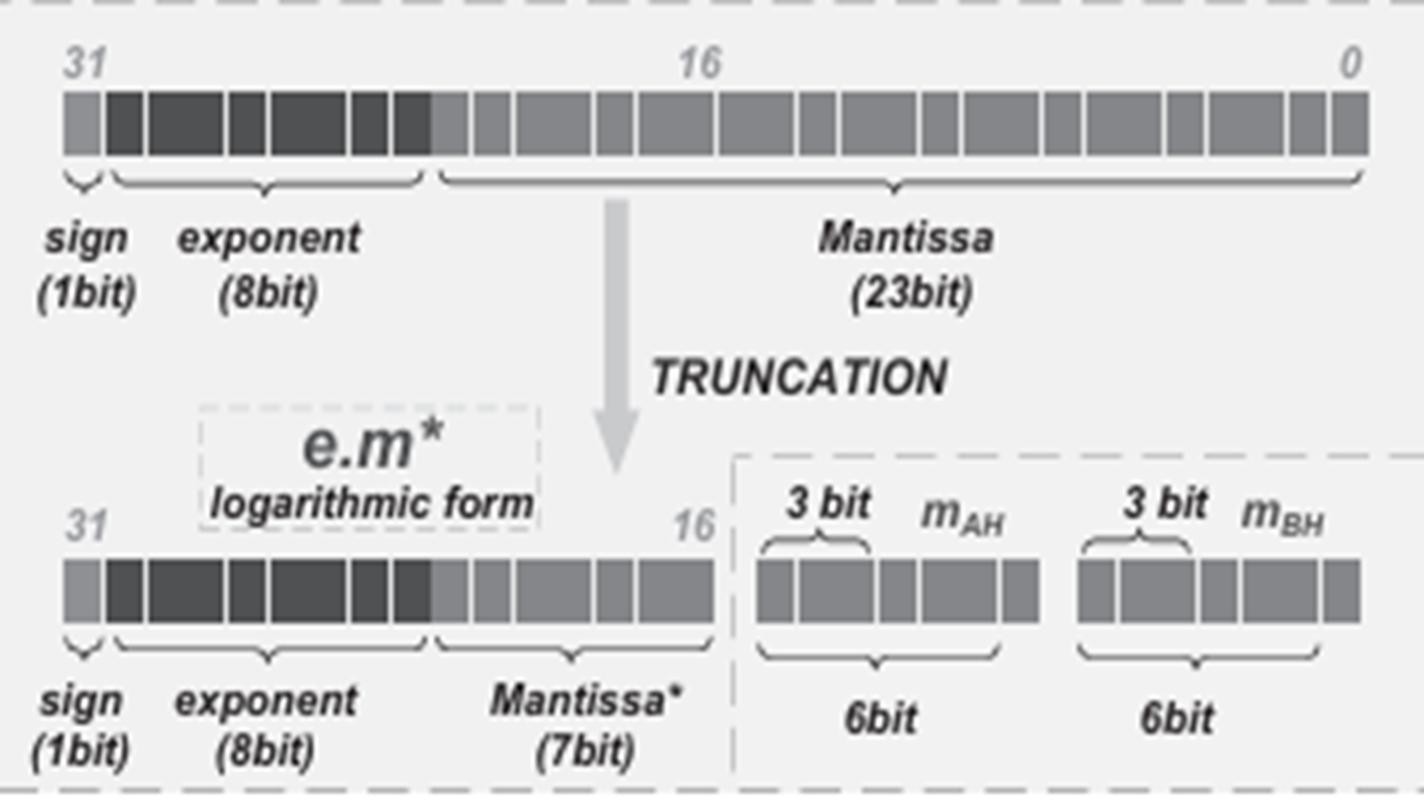


Figure -4 IEEE-754 Floating Point Representation

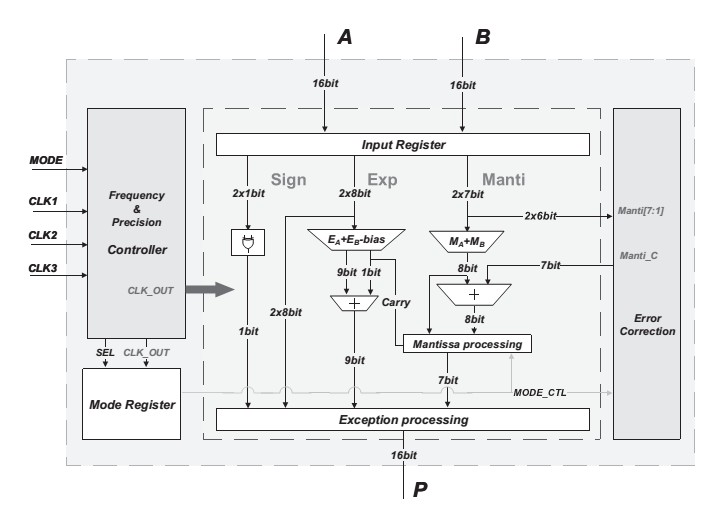
IEEE-754 is the standard for representing floating-point numbers in binary format. It defines formats for both single-precision (32-bit) and double-precision (64-bit) numbers and is widely used in computer systems for accurate and efficient arithmetic.

IEEE-754 is a standard to represent real numbers in binary form.

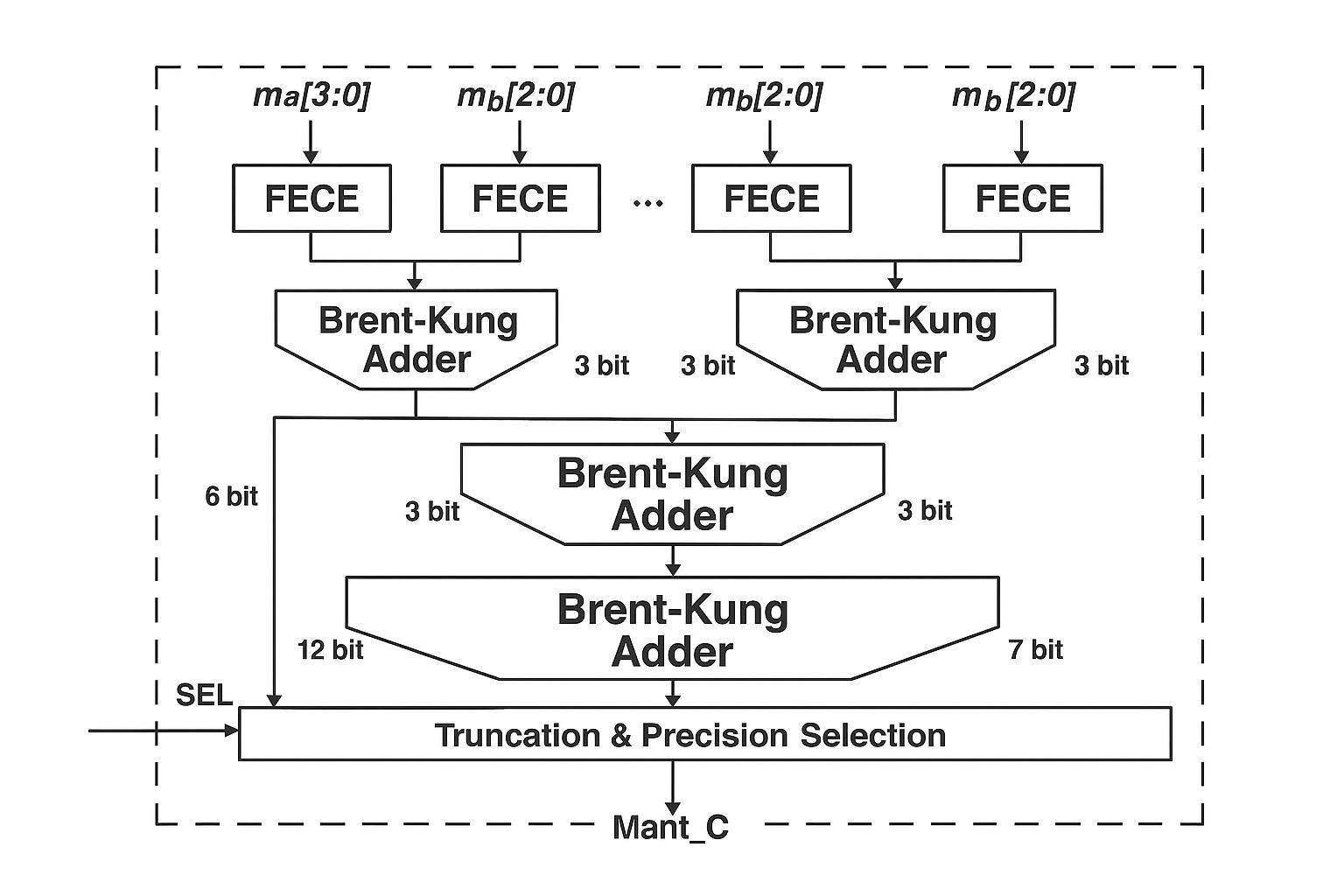
It has three parts: Sign bit, Exponent, and Mantissa.

* Sign bit: 0 for positive, 1 for negative
* Exponent: Adjusted using a bias (127 for single, 1023 for double)
* Mantissa: Stores the precision bits, with a hidden 1 before the point

## 4.3 Block Diagram



**FIGURE 5 : THE OVERALL ARCHITECTURE OF THE PROPOSED APPROXIMATE MULTIPLIER.** Figure. 5 illustrates the proposed approximate multiplier, composed of a multiplication core and a frequency and precision controller. The multiplication core simplifies floating-point multiplication into addition in the logarithmic domain, enhancing computational efficiency and reducing energy consumption. The frequency and precision controller allows seamless switching of three unrelated clock signals and sends the precision control signal to the multiplication core via the mode register, enabling runtime reconfiguration of precision and frequency. In addition, a compact error correction module is integrated in the multiplication core. The strategy of hardware reuse is adopted to implement two error correction modes at a small area cost. When the error correction module is not activated by the control signal, the addition result in the logarithmic domain is directly output (non-correction mode). When activated, the output of the error correction module is added to the result of the non-correction mode for processing.



**Figure 6 Error Correction Model:**

Inputs (Top of Diagram)

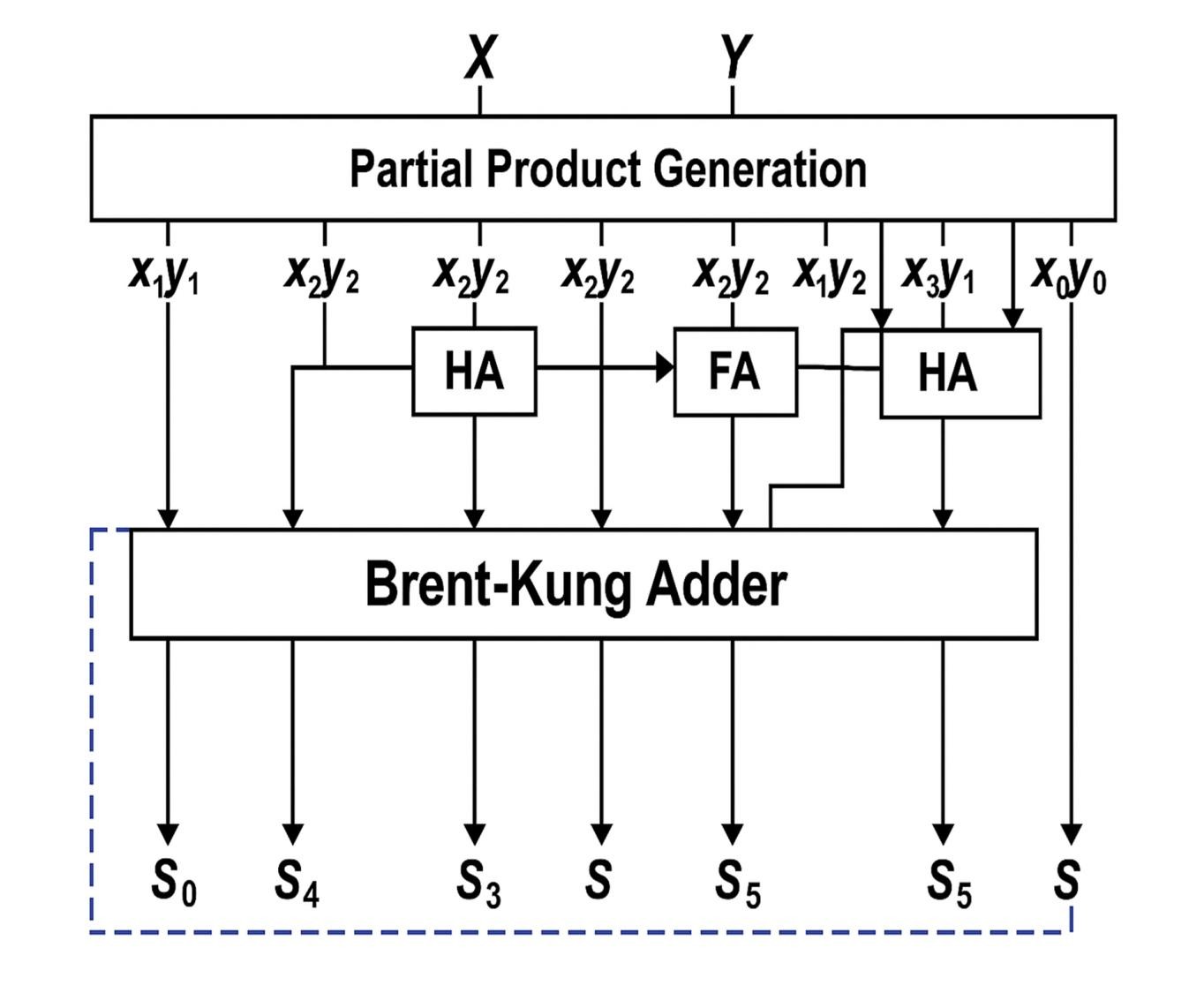
* The diagram shows inputs labeled as ma3:03:03:0 and multiple instances of mb2:02:02:0 (or something similar).
* These bit-slice names suggest that they’re portions of a larger operand, possibly partial mantissa bits if this is a floating-point datapath, or partial product terms in a multiplier design. FECE Blocks
* Each of the blocks at the top is labeled FECE. The exact meaning of “FECE” is not stated in the diagram itself, but these blocks typically act as partial-sum or partial-product generators in arithmetic logic (for instance, in a floating-point multiply or fused-add design).
* Because they feed into adders (Brent–Kung Adders), each FECE might be generating sums or partial combinations of bits that need to be added together.

Parallel Brent–Kung Adders

* Following the FECE blocks, the diagram shows multiple Brent–Kung Adders, each handling some of the bits resulting from the preceding FECE outputs.
* A Brent–Kung Adder is a well-known parallel prefix adder design that provides a good balance of gate depth and transistor count. It computes carries in a tree-like structure, then propagates sum bits in a relatively efficient way. Hierarchical Organization of Adders
* The diagram shows that there are multiple smaller Brent–Kung Adders producing, for instance, 3-bit or 6-bit sums. Then these partial sums appear to feed into another Brent– Kung Adder of possibly larger width (labeled 12-bit in the figure).
* This hierarchical structure is common in hardware designs that must handle partial sums of different sizes before producing a final result. The final wide adder likely combines these partial sums into a single 12-bit (or similarly sized) result. Truncation & Precision Selection
* After the final Brent–Kung Adder, there is a block labeled Truncation & Precision Selection.
* This suggests that the design can output results of varying precision (e.g., 8-bit, 10-bit, 12-bit, etc.) or possibly handle different floating-point formats (e.g., single vs. half precision).
* An input labeled SEL likely controls which mode or precision the datapath should apply (e.g., rounding, truncation, or partial bits masked out).

Output (Mant\_C)

* The final output is labeled Mant\_C, which strongly hints that this is the “mantissa computation” block of a floating-point unit. “C” might stand for “result” or might represent a specific pipeline stage name.
* In typical floating-point pipelines, once the significand (mantissa) is computed, it proceeds to normalization, rounding, or other finishing steps before becoming the final floating-point result



**Figure – 7 General Multiplication of Brunt-Kung Adder:**

## 4.4Advantages

* Dynamic precision control allows real-time adjustments for optimized accuracy and energy consumption, enabling a balance between computational efficiency and power savings.
* The frequency and precision controller provides adaptive trade-offs, allowing for seamless transitions between high accuracy and low energy consumption, optimizing performance for different applications.
* Logic size reduction is achieved using the Brent-Kung Adder, which minimizes the overall hardware footprint, leading to significant area savings and reducing fabrication costs. The error correction module provides tunable accuracy, ensuring performance adaptability based on specific application needs, making the design highly flexible for various workloads.
* The hardware implementation ensures low complexity and high throughput, making it feasible for real-time applications that demand efficient resource utilization.
* The combination of Brent-Kung Adder and approximate computing principles provides an optimal balance between energy savings and computational accuracy, making the design suitable for next-generation AI applications.
* The design methodology can be extended to other arithmetic operations, allowing for broader applications in digital signal processing, image processing, and neural network computations.
* Power Efficient: With reduced gate count and a less complex wiring structure, the BrentKung adder consumes less power than more complex designs.
* Moderate Speed: While not the fastest parallel prefix adder, the logarithmic depth ensures that it is faster than simpler designs like ripple-carry adders, striking a good balance between speed and hardware cost.
* The reduction in critical path delay improves overall system performance by optimizing the timing of operations, resulting in a more efficient computation pipeline.

# CHAPTER-5

**SOFTWARE TOOL**

MODELSIM - ALTRA

**Assumptions**

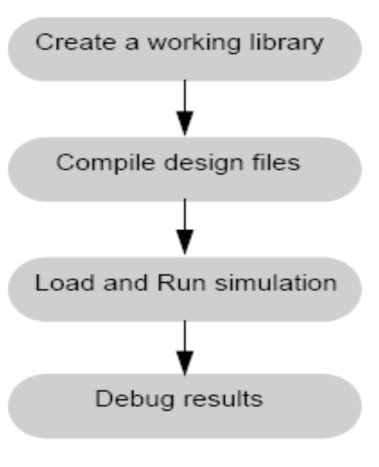
I assume that you are familiar with the use of your operating system. You should also be familiar with the window management functions of your graphic interface: Open Windows, OSF/Motif, CDE, KDE, GNOME, or Microsoft Windows 2000/XP. We also assume that you have a working knowledge of the language in which your design and/or test bench is written (i.e., VHDL, Verilog, etc.). Although ModelSim™ is an excellent tool to use while learning HDL concepts and practices, this document is not written to support that goal.

**ModelSim Introduction**

ModelSim is a verification and simulation tool for VHDL, Verilog, System Verilog, and mixed language designs. This lesson provides a brief conceptual overview of the ModelSim simulation environment. It is divided into four topics, which you will learn more about in subsequent lessons.

**Basic Simulation Flow**

The following diagram shows the basic steps for simulating a design in ModelSim.



**FIGURE 8: BASIC SIMULATION FLOW**

**Basic Simulation Flow - Overview Lab [Creating the Working Library]:**

In ModelSim, all designs are compiled into a library. You typically start a new simulation in ModelSim by creating a working library called "work," which is the default library name used by the compiler as the default destination for compiled design units.

**Compiling Your Design**

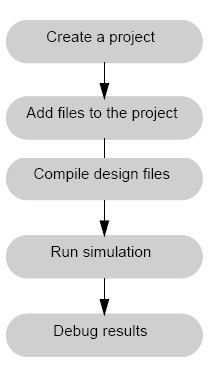
After creating the working library, and compile your design units into it. The ModelSim library format is compatible across all supported platforms. Its can simulate your design on any platform without having to recompile your design. Loading the Simulator with Your Design and Running the Simulation with the design compiled, load the simulator with your design by invoking the simulator on a top-level module (Verilog) or a configuration or entity/architecture pair (VHDL).

**Debugging Your Results**

If you don’t get the results you expect, you can use ModelSim’s robust debugging environment to track down the cause of the problem.

**Project Flow**

A project is a collection mechanism for an HDL design under specification or test. Even thoughyou don’t have to use projects in ModelSim, they may ease interaction with the tool and are useful for organizing files and specifying simulation settings. The following diagram shows the basic steps for simulating a design within a ModelSim project.



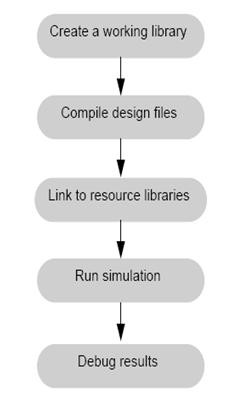
**FIGURE 9: PROJECT FLOW**

As you can see, the flow is similar to the basic simulation flow. However, there are two important differences:

* Do not have to create a working library in the project flow; it is done for you automatically.
* Projects are persistent. In other words, they will open every time you invoke ModelSim unless you specifically close them.

**Multiple Library Flow**

ModelSim uses libraries in two ways: 1) as a local working library that contains the compiled version of your design; 2) as a resource library. The contents of your working library will change as you update your design and recompile. A resource library is typically static and serves as a parts source for your design. It can create your own resource libraries, or they may be supplied by another design team or a third party (e.g., a silicon vendor). It specifies which resource libraries will be used when the design is compiled, and there are rules to specify in which order they are searched. A common example of using both a working library and a resource library is one where your gate-level design and test bench are compiled into the working library, and the design references gate-level models in a separate resource library. The diagram below shows the basic steps for simulating with multiple libraries.



**FIGURE 10 : MULTIPLE LIBRARY FLOW**

**Debugging Tools**

Model Sim offers numerous tools for debugging and analyzing your design. Several of these tools are covered in subsequent lessons, including:

* Using projects
* Working with multiple libraries
* Setting breakpoints and stepping through the source code
* Viewing waveforms and measuring time
* Viewing and initializing memories
* Creating stimulus with the Waveform Editor
* Automating simulation

**Basic Simulation**

**Introduction**

In this lesson you will go step-by-step through the basic simulation flow:

1. Create the Working Design Library
2. Compile the Design Units
3. Load the Design
4. Run the Simulation

**Design Files for this Lesson**

The sample design for this lesson is a simple 8-bit, binary up-counter with an associated test bench. The pathnames are as follows:

**Verilog** – <install\_dir>/examples/tutorials/verilog/basicSimulation/counter.v and tcounter.v

**VHDL** – <install\_dir>/examples/tutorials/vhdl/basicSimulation/counter.vhd and tcounter.vhd

This lesson uses the Verilog files *counter.v* and *tcounter.v*. If you have a VHDL license, use

*Counter.vhd* and *tcounter.vhd* instead. Or, if you have a mixed license, feel free to use the

Verilog test bench with the VHDL counter or vice versa.

Compile HDL Libraries through Project Navigator

**Setting up Environment Variables**

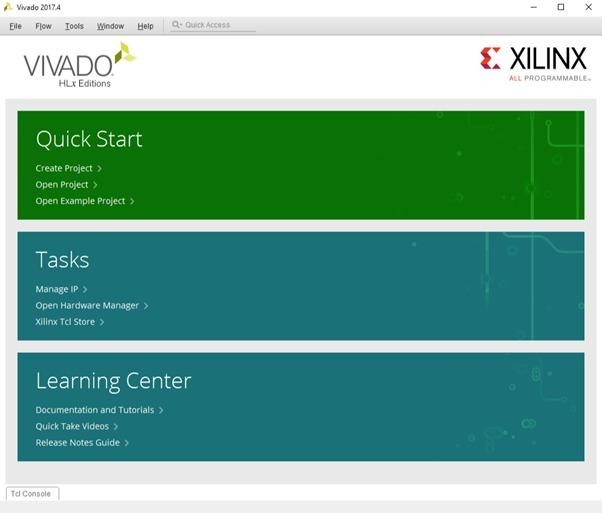
Before we discuss the individual methods, let us first understand how ModelSim accesses

VIVADO

Step 1: Create a Vivado Project

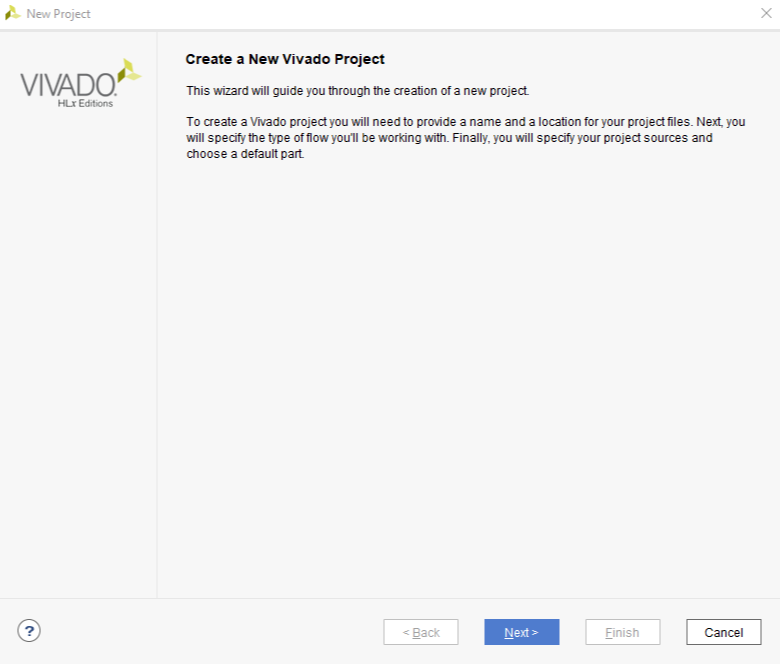
Vivado “projects” are directory structures that contain all the files needed by a particular design. Some of these files are user-created source files that describe and constrain the design, but many others are system files created by Vivado to manage the design, simulation, and implementation of projects. In a typical design, you will only be concerned with the user-created source files. But, in the future, if you need more information about your design, or if you need more precise control over certain implementation details, you can access the other files as well.

When setting up a project in Vivado, you must give the project a unique name, choose a location to store all the project files, specify the type of project you are creating, add any pre-existing source files or constraints files (you might add existing sources if you are modifying an earlier design, but if you are creating a new design from scratch, you won’t add any existing files – you haven’t written them yet), and finally, select which physical chip you are designing for. These steps are illustrated below.



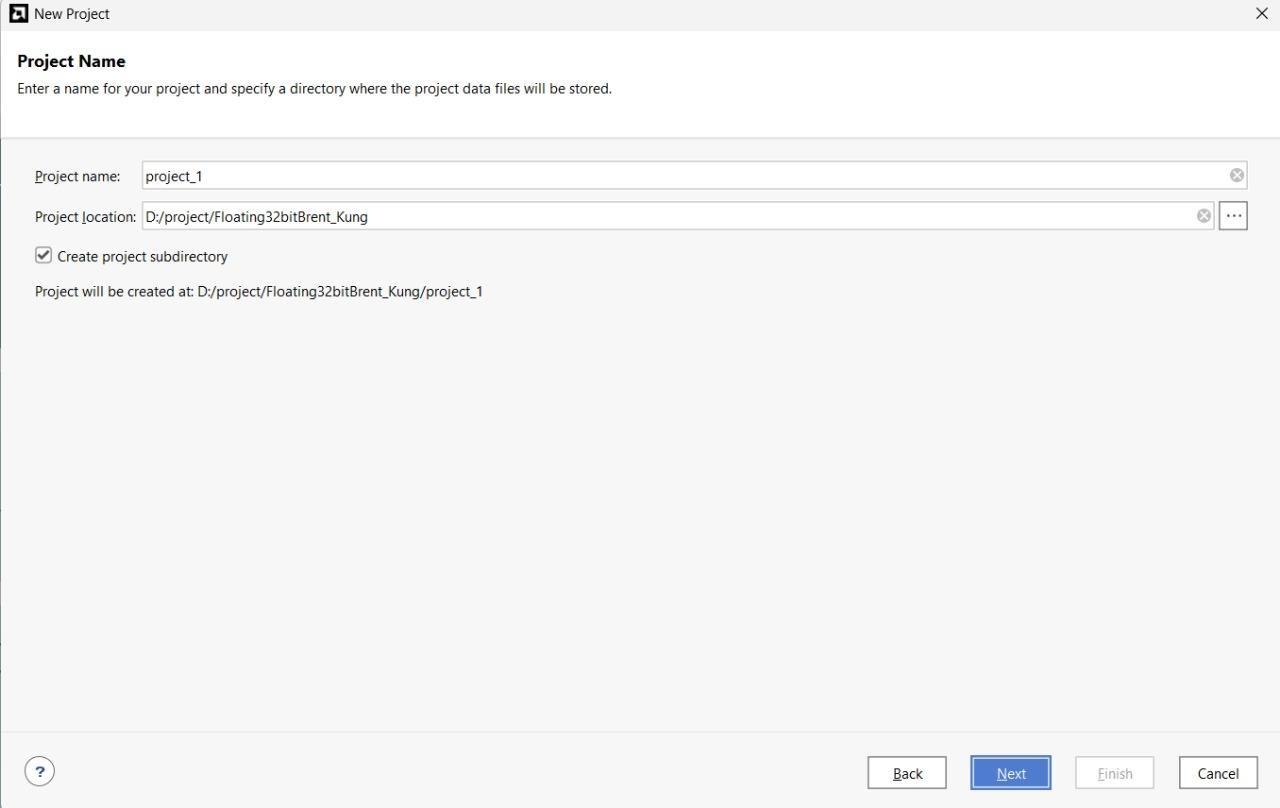
*Open Create Project Dialog*

Click on “Create Project” in the Quick Start panel. This will open the New Project dialog as shown in above Figure. Click Next to continue.



Create Project Dialog

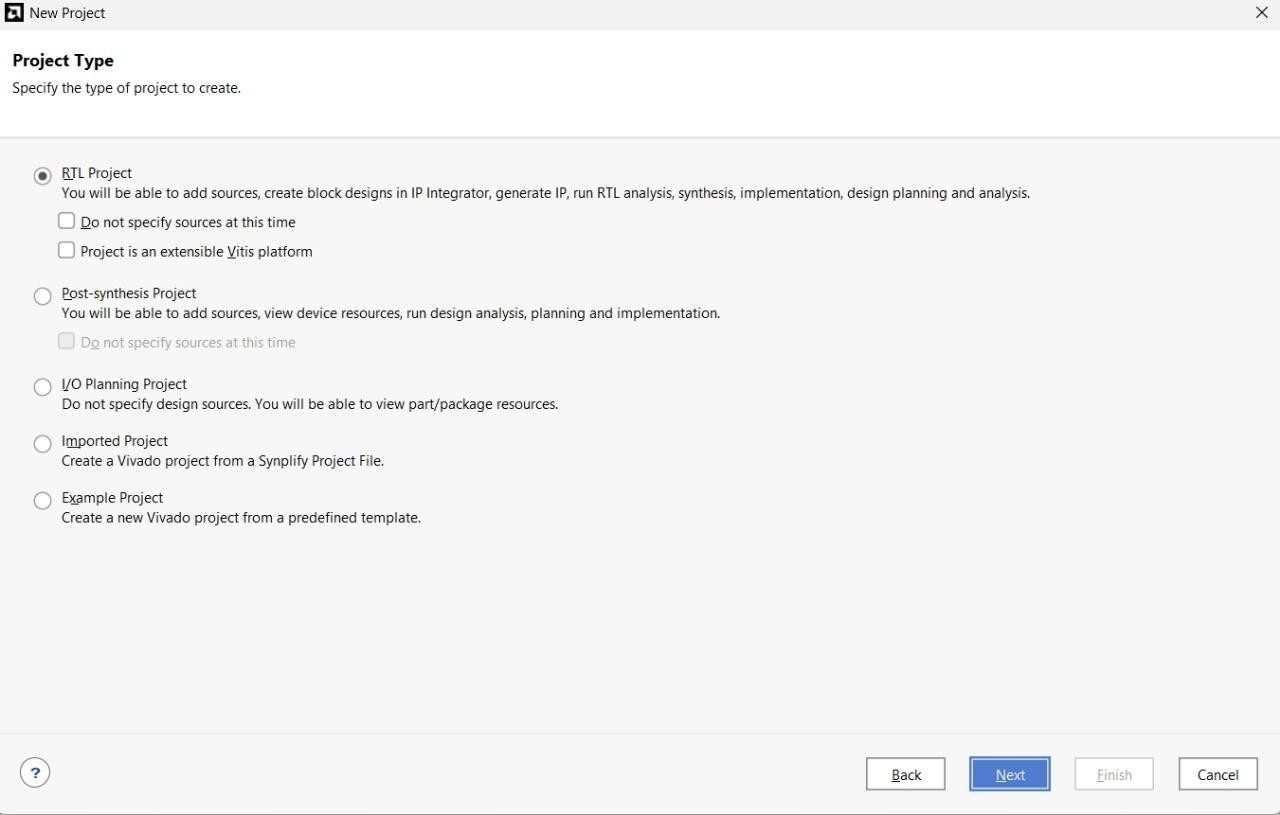
*Set Project Name and Location* Enter a name for the project. In the figure, the project name is “project\_1”, which isn’t a particulary useful name. It’s usually a good idea to make the project name more descriptive, so you can more readily identify your designs in the future. For example, if you design a seven-segment controller, you might call the project “seven segment controller”. For projects related to coursework, you might include the course name and project number - for example, “Floating32bitBrent\_kung”. You should avoid having spaces in the project name or location, because spaces can cause certain tools to fail.



Enter Project Name

*Select Project Type*

The “project type” configures certain design tools and the IDE appearance based on the type of project you are intending to create. Most of the time, and for all Real Digital courses, you will choose “RTL Project” to configure the tools for the creation of a new design. (RTL stands for Register Transfer Language, which is a term sometimes used to mean a hardware design language like Verilog).

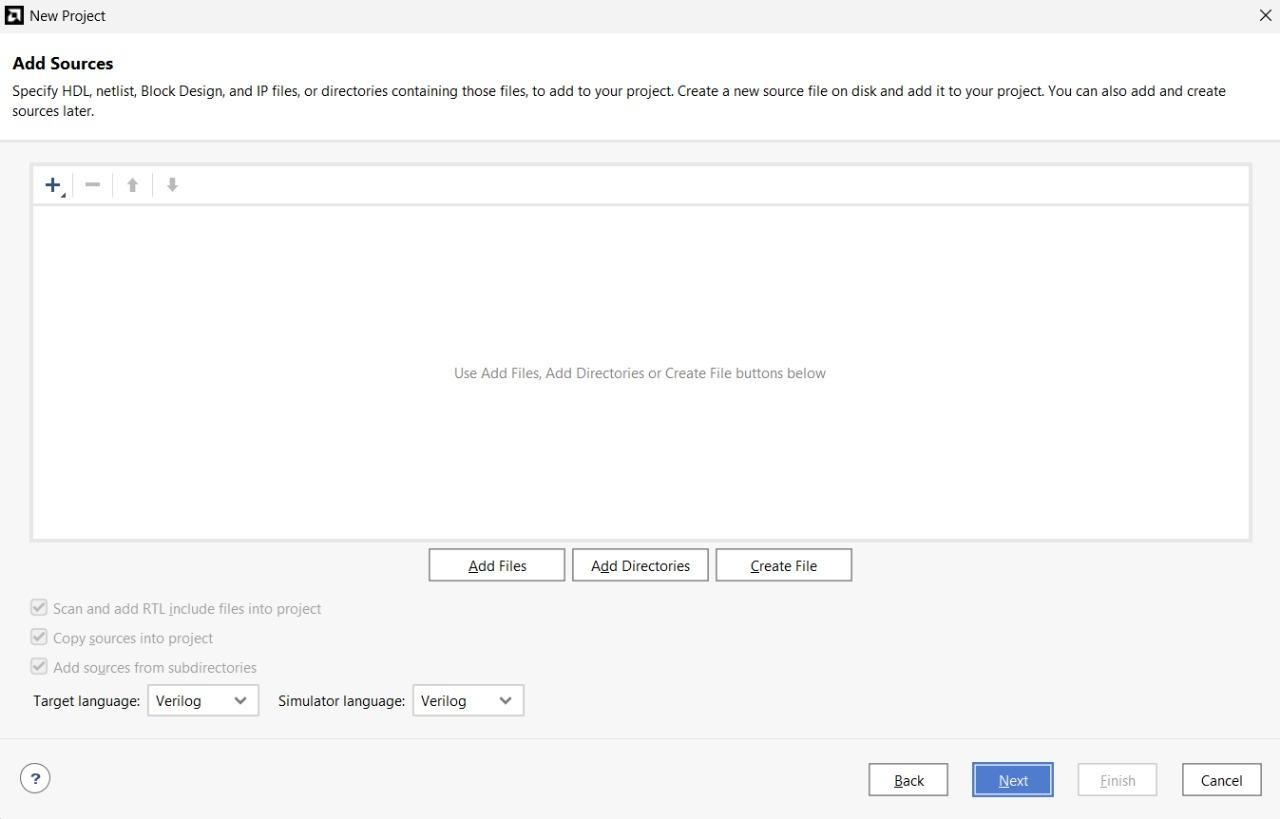


Select Project Type

*Add Existing Sources*

In a typical new or early-stage design, you won’t add any existing sources because you haven’t created them yet. But as you complete more designs and build up a library of previously completed and known good designs, you may elect to add sources and them use them in a new design.

For now, there are no existing sources to add, so just click Next.

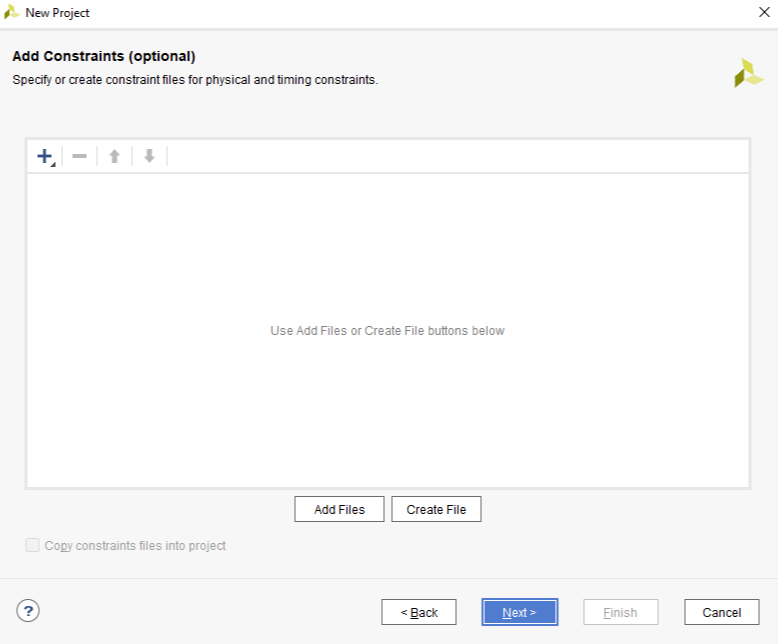


. Add Sources

*Add Constraints*

Constraint files provide information about the physical implementation of the design. They are created by the user, and used by the synthesizer. Constraints are parameters that specify certain details about the design. As examples, some constraints identify which physical pins on the chip are to be connected to which named circuit nodes in your design; some constraints setup various physical attributes of the chip, like I/O pin drive strength (high or low current); and some constraints identify physical locations of certain circuit components.

The Xilinx Design Constraints (.xdc filetpye) is the file format used for describing design constraints, and you need to create an .xdc file in order to synthesize your designs for a Real Digital board. Later in this tutorial, you will create a constraints file to identify which named circuit nodes must be connected to which physical pins. But for now, you have no existing constraints file to add, so you can simply click next.

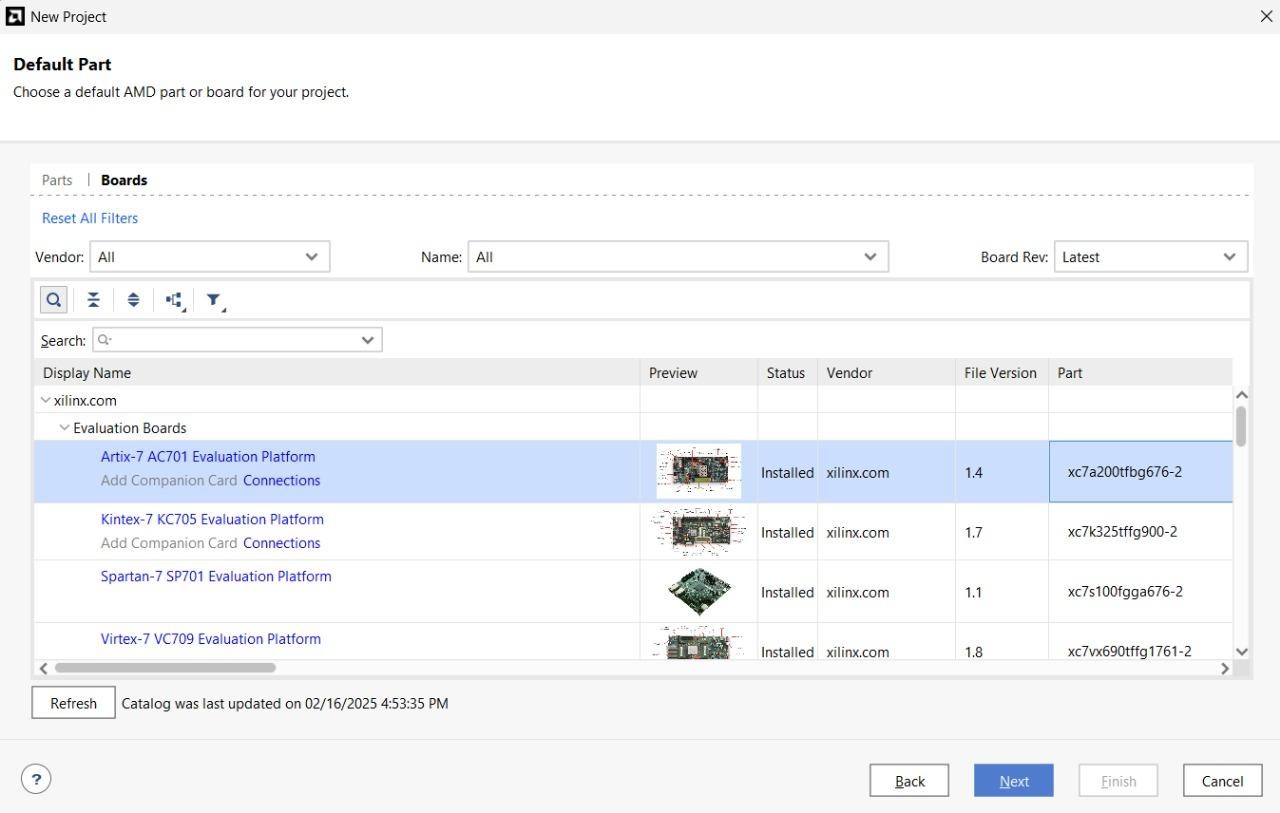


Add Constraint Files

*Select Parts*

Xilinx produces many different parts, and the synthesizer needs to know exactly what part you are using so it can produce the correct programming file. To specify the correct part, you need to know the device family and package, and less critically, the speed and temperature grades (the speed and temperature grades only affect special-purpose simulation results, and they have no effect on the synthesizer’s ability to produce accurate circuits). You must choose the appropriate part for the device installed on your board.

Select the board **xc7a200tfbg676-2**

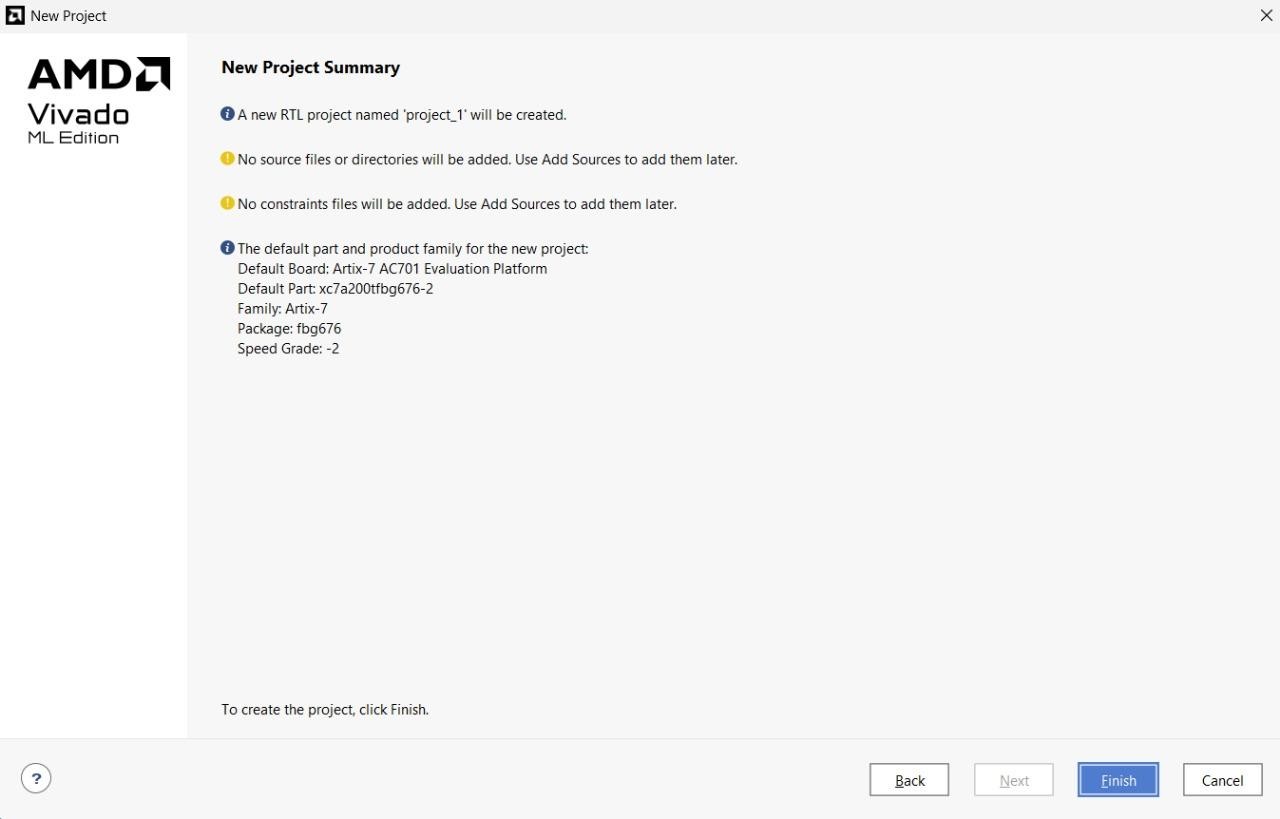


|  |  |
| --- | --- |
| **Part Number** | **xc7a200tfbg676-2** |
| **Family** | **Artix-7** |
| **Package** | **fbg676** |
| **Speed Grade** | **-2** |

**Check Project Configuration Summary**

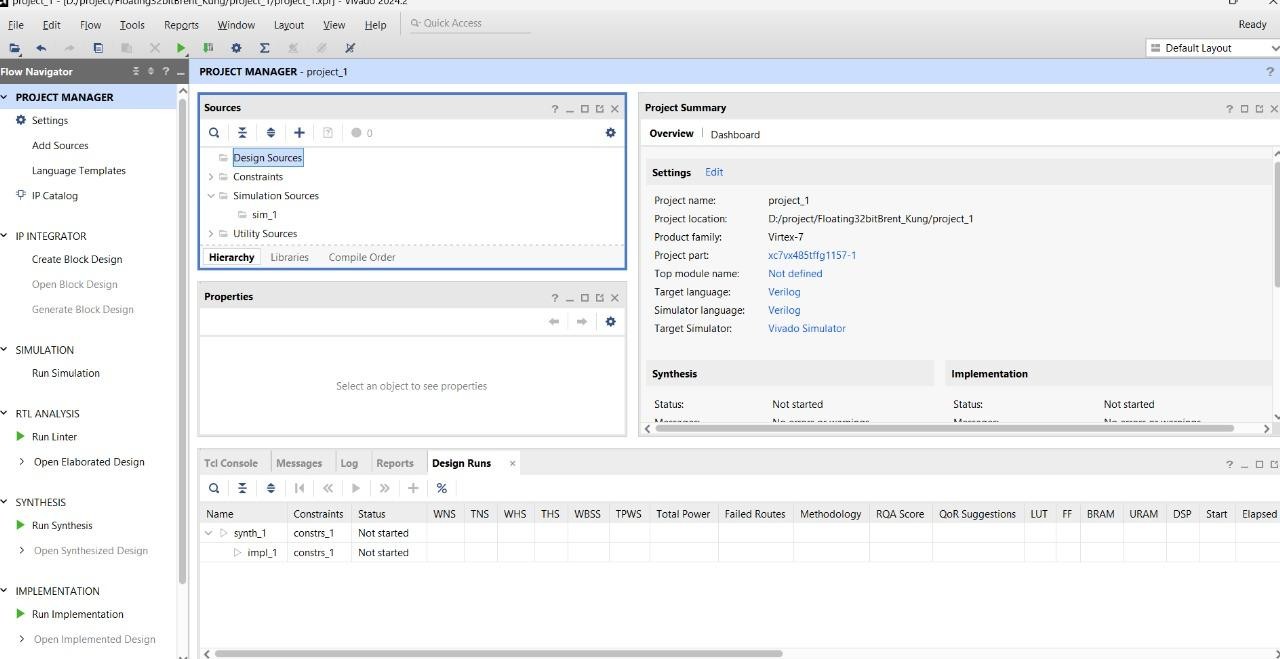
On the last page of the Create Project Wizard a summary of the project configuration is shown. Verify all the information in the summary is correct, and in particular make sure the correct FPGA part is selected. If anything is incorrect, click back and fix it; otheriwse, click Finish to finish creating an empty project.

And Create Project Summary



**Vivado Project Window**

After you have finished with the Create Project Wizard, the main IDE window will be displayed. This is the main “working” window where you enter and simulate your Verilog code, launch the synthesizer, and program your board. The left-most pane is the flow navigator that shows all the current files in the project, and the processes you can run on those files. To the right of the flow navigator is the project manager window where you enter source code, view simulation data, and interact with your design. The console window across the bottom shows a running status log. Over the next few projects, you will interact with all of the panels.



**Step-2: Edit The Project - Create source files**

All projects require at least two types of source files – an HDL file (Verilog or VHDL) to describe the circuit, and a constraints file to provide the synthesizer with the information it needs to map your circuit into the target chip.

This tutorial presents the steps required to implement a Verilog circuit on your Real Digital board: first, a Verilog source file is created to define the circuits behavior (again, for this tutorial, you can simply copy or download the completed file rather than typing it); second, a constraints files is created to define how the Verilog circuit is mapped into the Xiling logic device (again, copied or downloaded for this tutorial); third, the Verilog source file and constraints file are synthesized into a “.bit” file that can be programmed onto your board; and fourth, the device is configured with the circuit.

After the Verilog source file is created, it can be directly simulated. Simulation (discussed in more detail later) lets you work with a computer model of a circuit, so you can check its behavior before taking the time to implement it in a physical device. The simulator lets you drive all the circuit inputs with varying patterns over time, and to check that the outputs behave as expected under all conditions.

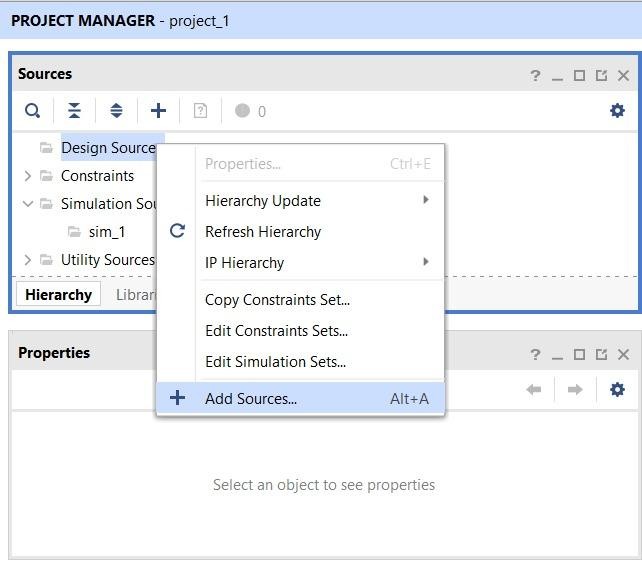
After the constraint file is created, the design can be synthesized. The synthesis process translates Verilog source code into logical operations, and it uses the constraints file to map the logical operations into a given chip. In particular (for our needs here), the constraints file defines which Verilog circuit nodes are attached to which pins on the Xilinx chip package, and therefore, which circuit nodes are attached to which physical devices on your board. The synthesis process creates a “.bit” file that can be directly programmed into the Xilinx chip.

In this first tutorial, the Verilog and constraint source files are provided for you. Instead of creating them yourself as would normally be the case, you can simply copy them into empty source files, or download them and include them in your project directly. In later designs, you will create these files yourself.

**Design Sources**

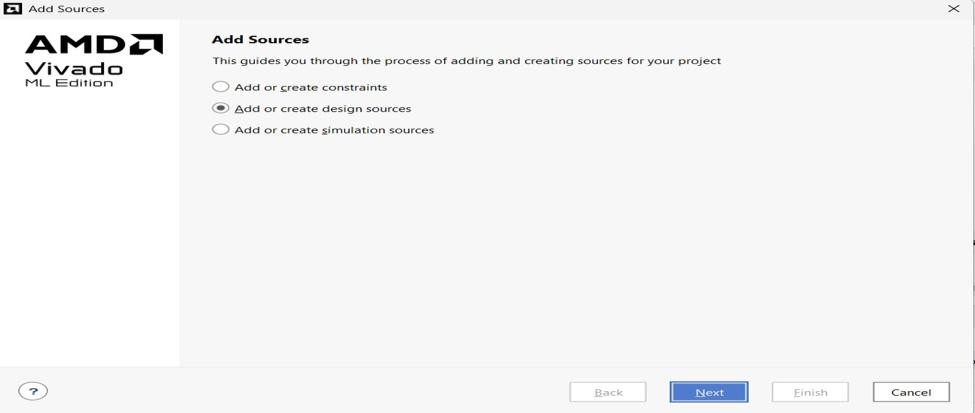
There are many ways to define a logic circuit, and many types of source files including VHDL, Verilog, EDIF and NGC netlists, DCP checkpoint files, TCL scripts, System C files, and many others. We will use the Verilog language in this course, and introduce it gradually over the first several projects. For now, you can get familiar with some of the basic concepts by reading the following.

**VERILOG HDL: THE FIRST EXAMPLE**



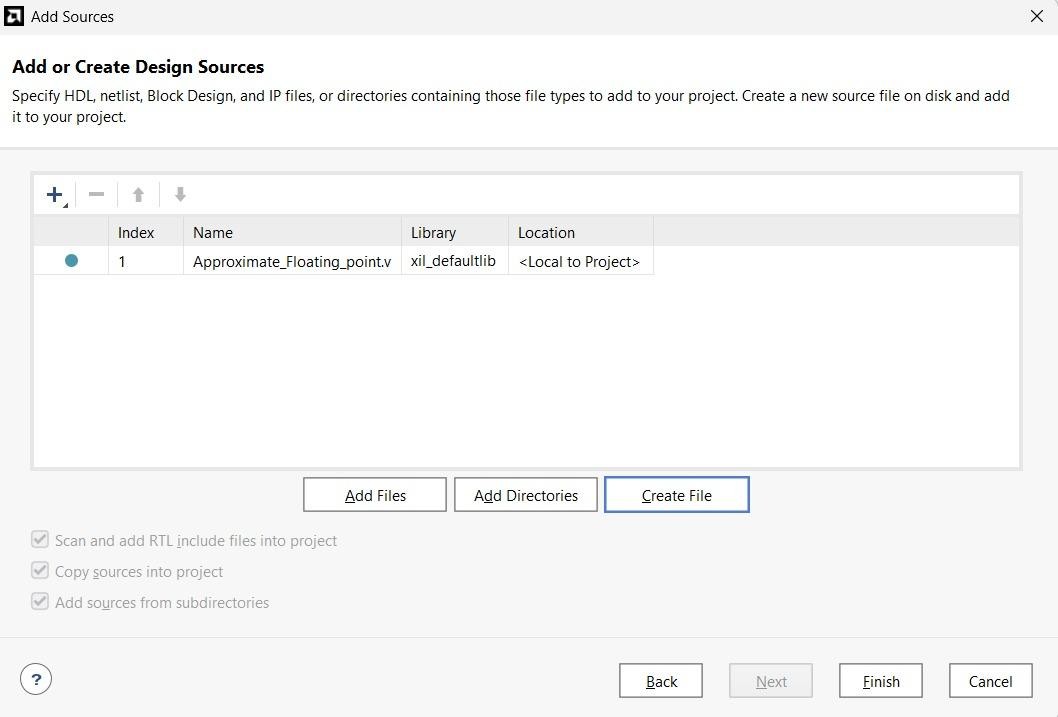
To create a Verilog source file for your project, right-click on “Design Sources” in the Sources panel, and select Add Sources. The Add Sources dialog box will appear as shown –

select “Add or create design sources” and click next.



Add or create design sources using Add Source Dialog

In the Add or Create Design Sources dialog, click on Create File, enter project1\_demo as filename, and click OK. The newly created file will appear in the list as shown. Click Finish to move to the next step.



Create Design Source File

Skip the Define Module dialog by clicking OK to continue.

And click to Finish.

**Simulation Library Compilation Wizard**

This method is highly recommended because library compilation can be performed for more than one device family and/or language.

1. Open the Simulation Library Compilation Wizard. This can be accessed from Start Menu→Programs→Xilinx ISE Design Suite 13.x→ISE Design Tools→32-bit Tools

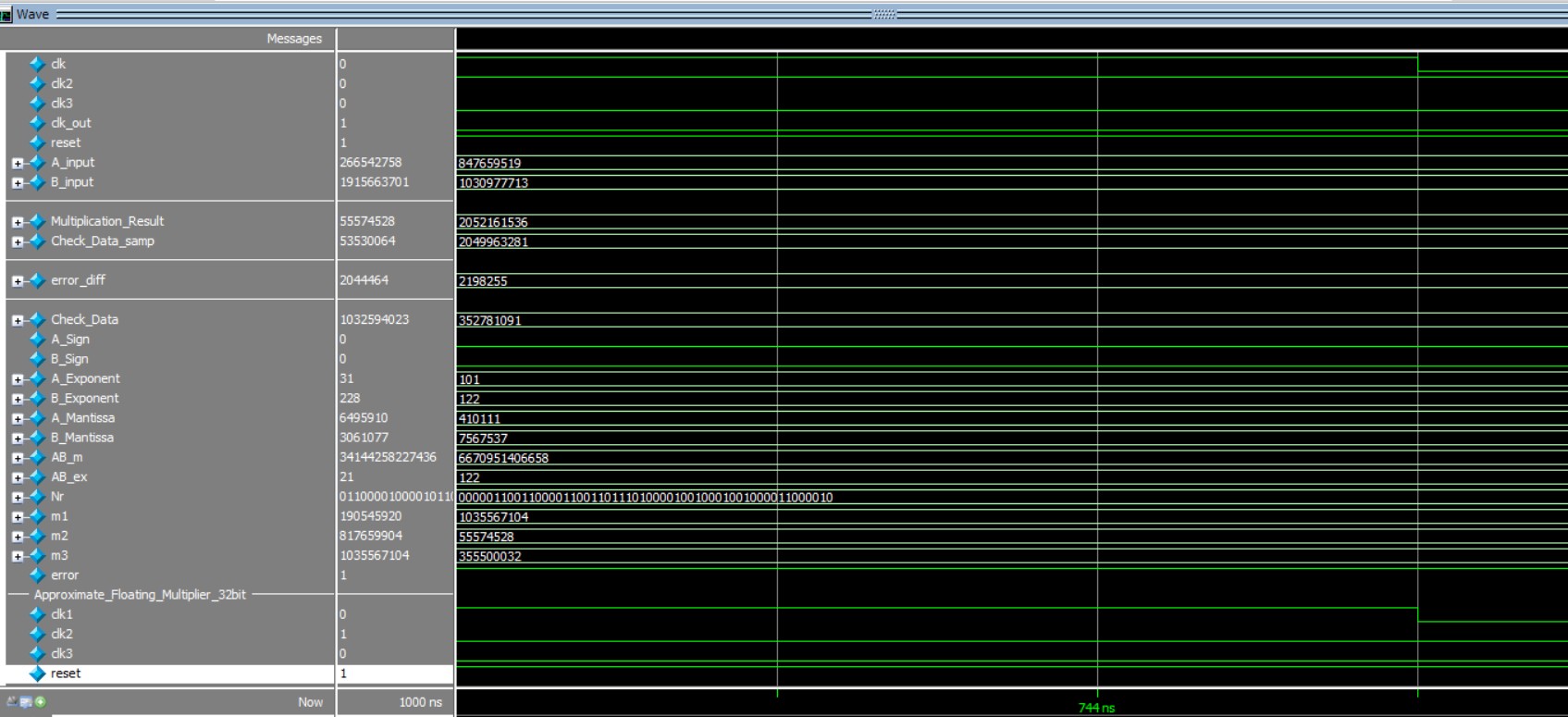
**Note:** If you are using 64-bit version of ModelSim use the Simulation Library Compilation Wizard from 64-bit Tools. ModelSim PE Student Version 10.x is only available in the 32-bit version.

1. The Select Simulator window opens up. Select the appropriate simulator (For Student Edition select ModelSim PE); enter c:\modeltech64 10.0cnwin64 for executable location, compxlib.cfg for Compxlib Configuration File and compxlib.log for Compxlib Log File.
2. Next select the HDL used for simulation. If you are unsure select Both VHDL and Verilog. However, this will increase the compilation time and the disk space required.
3. Then select all the device families that you will be working with. Again the more number of devices, more the compilation time and the disk space required. Remember that you can always run the compilation wizard at a later time for additional devices.
4. The next window is for selecting libraries for Functional and Timing Simulation. Different libraries are required for different types of simulation (behavioral, post-route, etc.). We suggest that you select All Libraries as the default option. Interested users can refer to Chapter 6 of the Xilinx Synthesis and Simulation Design Guide for additional information.
5. Finally the window for Output directory for compiled libraries is shown. We suggest leaving the default values that Xilinx picks. Then select Launch Compile Process.
6. Be patient as the compilation can take a long time depending on the options that you have chosen.
7. The compile process may have contained a lot of warnings but should be error-free. We have not explored the reasons behind these warnings, but they do not appear to affect the simulation of any of our designs.
8. Once the process is completed, open c:\modeltech64 10.0cnmodelsim.ini and verifies if there are libraries pointing to the output directory entered in step 6. This will happen only if you have set the environment

# 

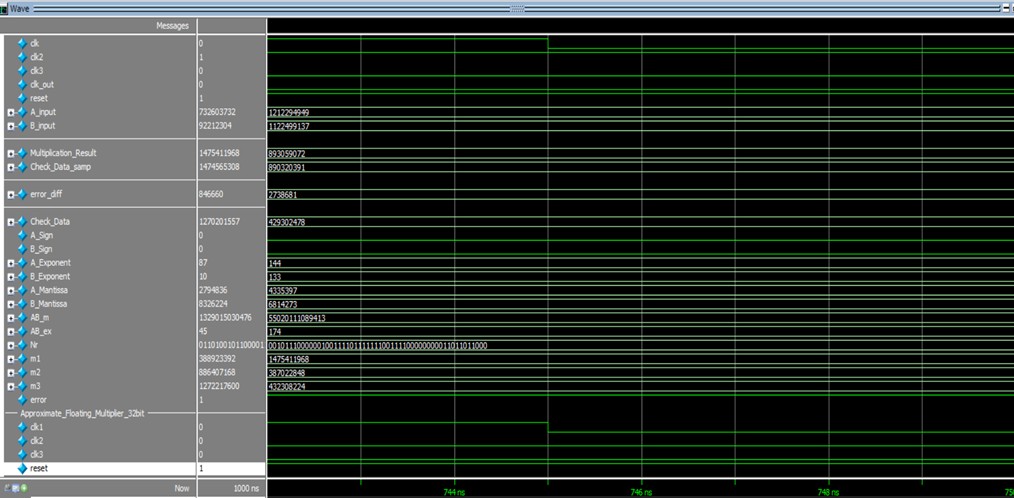
# CHAPTER-6

# Results



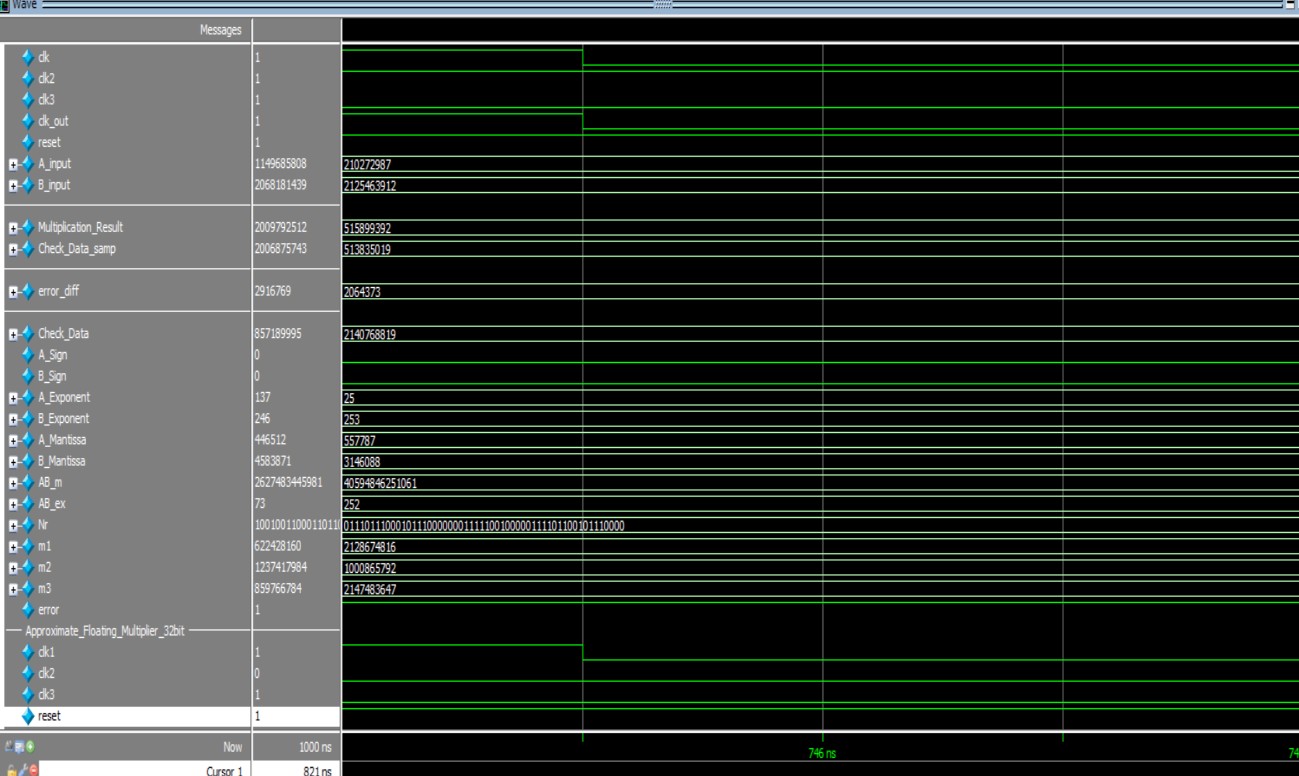
Low Precision Mode:

The waveform illustrates the simulation of a 32-bit approximate floating-point multiplier operating in low precision mode, utilizing a Brent-Kung Adder for efficient addition. The design achieves reduced power and area while maintaining acceptable accuracy, as indicated by the minimal error-diff between the computed and reference results.



High precision Mode

The waveform illustrates the high precision mode operation of the 32-bit approximate floating-point multiplier utilizing a Brent-Kung Adder for optimized addition. In this mode, higher mantissa accuracy is preserved, resulting in a significantly reduced error\_diff, demonstrating improved computational precision while maintaining efficient performance and hardware utilization.

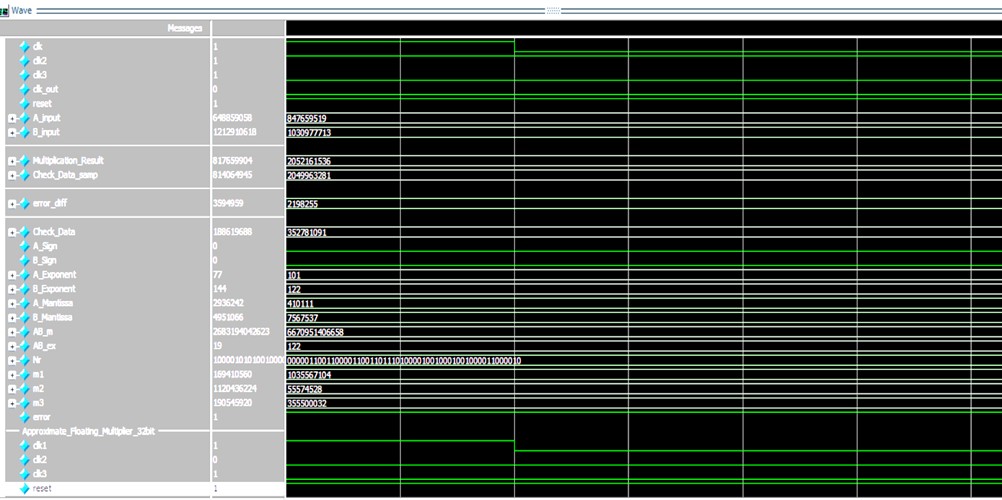


No Correction Mode

The waveform shows the **no correction mode** of a 32-bit approximate floating-point multiplier utilizing the **Brent-Kung Adder** for fast and area-efficient addition. In this mode, without error compensation logic, the design prioritizes performance and low power, leading to a noticeable increase in error-diff highlighting the trade-off between accuracy and energy efficiency in approximation-based hardware.

Existing Methodology :

Approximate Floating-point With CLA Adder:



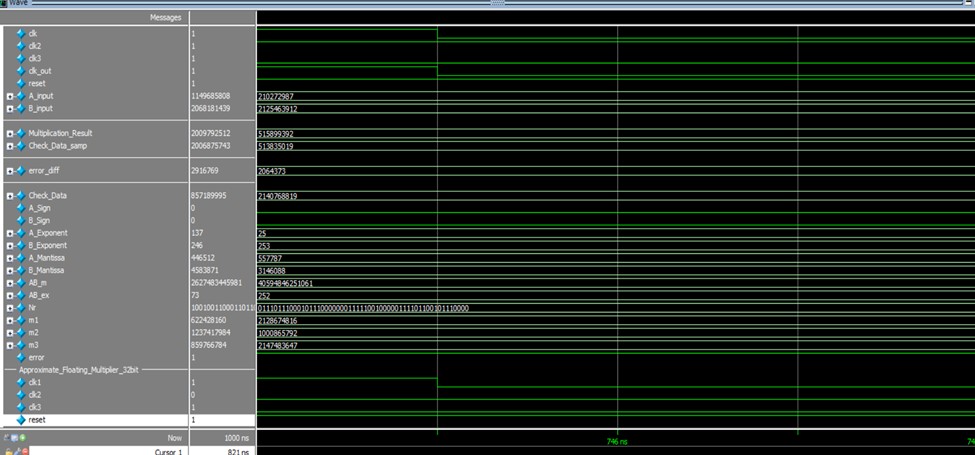
Low-Precision Mode :

The waveform represents the Low-precision mode of the 32-bit approximate floating-point multiplier, where no error compensation is applied during computation. Using a Carry Lookahead Adder (CLA) for fast addition, the design prioritizes speed and energy efficiency. As a result, a balance error-diff is observed, reflecting the trade-off made by omitting correction logic in favor of reduced hardware complexity and power consumption.



High-Precision Mode:

The waveform represents the Low-precision mode of the 32-bit approximate floating-point multiplier, where no error compensation is applied during computation. Using a Carry Lookahead Adder (CLA) for fast addition, the design prioritizes speed and energy efficiency. As a result, a low error-diff is observed, reflecting the trade-off made by omitting correction logic in favor of reduced hardware complexity and power consumption.



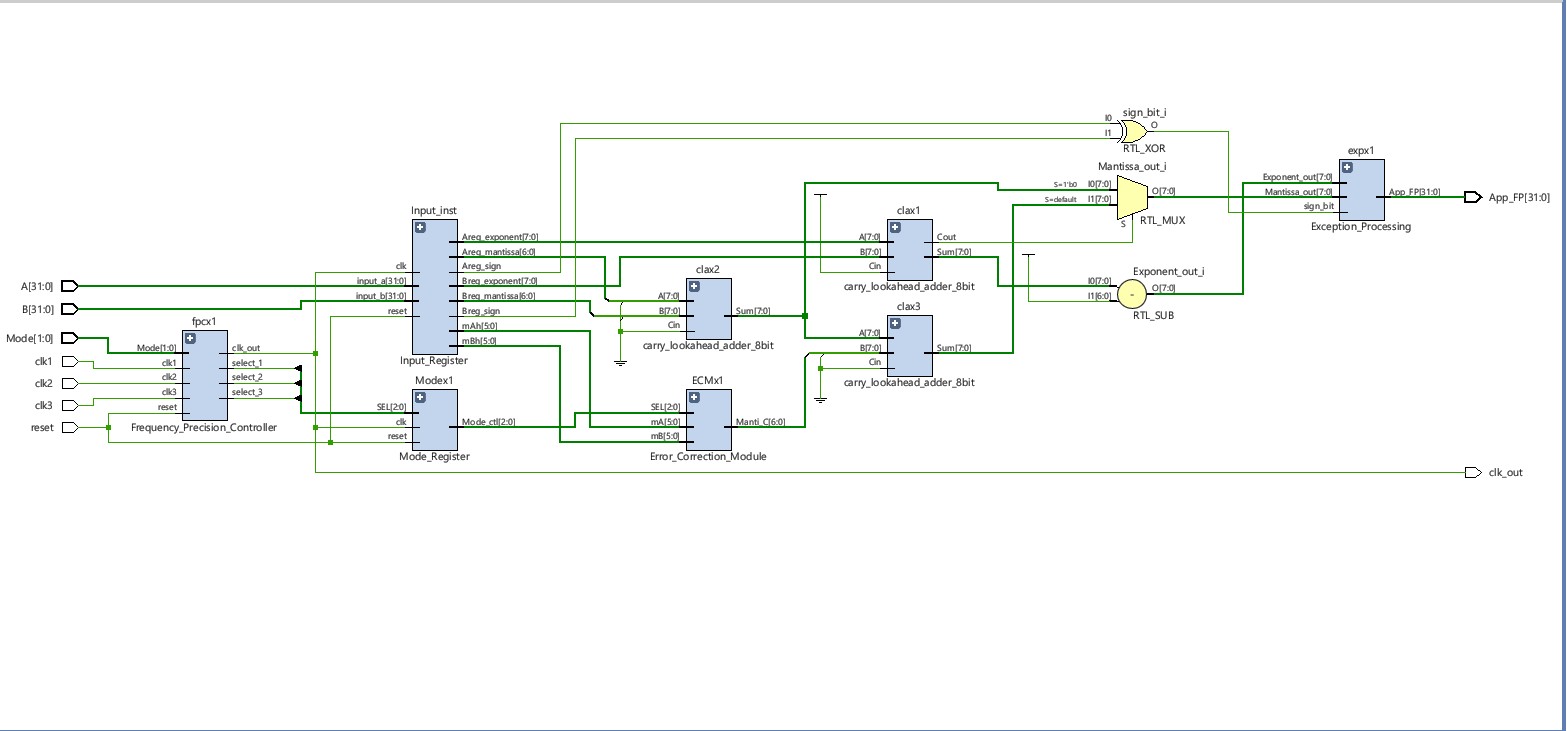
No Correction Mode

As a result, a high error-diff is observed, reflecting the trade-off made by omitting correction logic in favor of reduced hardware complexity and power consumption.

Implementation Results:

RTL View :

Approximate Floating-point with CLA Adder:



Approximate Floating-point with Brent-Kung Adder:

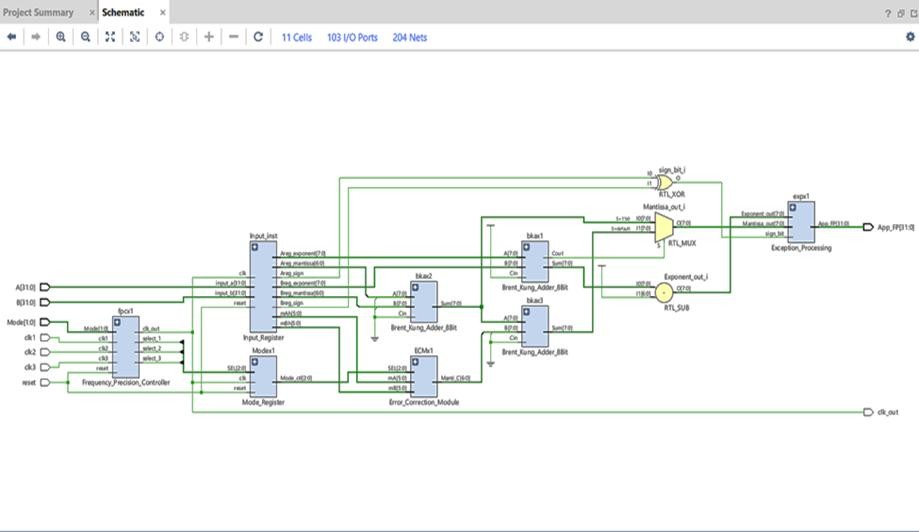


Table 1 : Comparison Between Floating-point Multiplier using CLA Adder and Floating-point Brent-Kung Adder

|  |  |  |
| --- | --- | --- |
|  | Floating-point Multiplier using CLA Adder | Floating-point  Multiplier using BrentKung Adder |
| Number of LUT ‘s | 78 | 41 |
| Number of FF | 41 | 41 |
| Number of IOB | 71 | 71 |
| Number of  BUFG’s | 4 | 4 |
| Power(W) | 13.023 W | 11.67 W |

The comparison table highlights the resource utilization and power consumption differences between a Floating-Point Multiplier implemented using a Carry Lookahead Adder and the one using a Brent-Kung Parallel Prefix Adder. The key observations from the table are as follows:

* Number of LUTs: The Brent-Kung Adder-based multiplier requires only 41 Look-Up Tables (LUTs), whereas the Carry Lookahead Adder-based design utilizes 78 LUTs. This demonstrates a significant reduction in logic utilization, making the Brent-Kung design more efficient in terms of area.
* Number of Flip-Flops (FF): Both designs utilize 41 flip-flops, indicating that the sequential elements required for storage remain the same.
* Number of IOBs: The number of Input/Output Blocks (IOBs) remains constant at 71 for both implementations, implying that external interfacing requirements do not change.
* Number of BUFGs: Both architectures use 4 global clock buffers (BUFG), suggesting that clock distribution overhead remains unchanged.
* Power Consumption: The Brent-Kung Adder-based multiplier achieves lower power consumption at 11.67W, compared to 13.023W for the Carry Lookahead Adder-based multiplier. This reduction in power usage makes the Brent-Kung Adder a more energyefficient choice, particularly for power-constrained applications. Overall, the comparison shows that replacing the Carry Lookahead Adder with a Brent-Kung Parallel Prefix Adder results in significant area savings (reduced LUT usage) and lower power

**CHAPTER-7**

# Conclusion And Future Scope

**Conclusion**

The design of the Approximate Floating-Point Multiplier using Brent-Kung Adder has been shown to provide high area and power efficiency along with reasonable levels of accuracy for other calculations. The dynamic reconfigurable precision control mechanism allows for speed, power, and accuracy adaptability, and the design proves to be highly flexible based on application requirements. By including the Brent-Kung Adder, the multiplier has reduced logic complexity and carry propagation optimization, further making it suitable for FPGA implementation. Experimental outcomes verify the performance of the presented design, with reduced power consumption and hardware overhead over traditional multipliers. This paper highlights the potential of approximate computing techniques in today's hardware design, particularly in energylimited environments such as edge computing and real-time computing. Future research can focus on optimizing error correction mechanisms and examining other optimization techniques to further improve performance without sacrificing much computational precision.

# Future Scope

* **Enhanced Approximation Techniques:** Future work can explore advanced approximation methods to further optimize power and area while maintaining acceptable accuracy levels.
* **Integration with AI Accelerators:** The proposed design can be integrated into AI and deep learning hardware accelerators for energy-efficient neural network processing.
* **Adaptive Precision Control:** Implementing dynamic precision scaling based on workload requirements can improve performance for applications requiring variable accuracy.
* **ASIC Implementation:** The design can be fabricated as an ASIC to evaluate its performance in real-world applications beyond FPGA-based implementations.

# References

1. K. Han, J. Choi, and K. Kang, "Approximate Computing for Deep Neural Networks: Energy Efficiency through Precision Trade-offs," IEEE Transactions on Neural Networks and Learning Systems, vol. 31, no. 12, 2020, pp. 5707-5719.
2. R. Zimmermann, "Brent-Kung Adder: An Efficient Architecture for Low-Area Arithmetic Circuits," IEEE Transactions on Computers, vol. 42, no. 2, 1993, pp. 259-269.
3. H. Jiang, J. Han, and F. Lombardi, "Error-Tolerant Floating-Point Arithmetic: Design and Application in Energy-Constrained Systems," IEEE Transactions on Computers, vol. 60, no. 2, 2011, pp. 87-96.
4. P. Judd, J. Albericio, and T. Hetherington, "Reconfigurable Clock Systems for Low-Power VLSI: Balancing Frequency and Precision," ACM/IEEE Design Automation Conference, 2016, pp. 123-130.
5. A. Shafique, M. Khalid, and M. Shafique, "Hardware-Efficient Approximate Multipliers: A Comparative Study," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 25, no. 10, 2017, pp. 1-11.
6. S. Gupta, P. Gupta, and S. Saxena, "Energy-Efficient Multipliers for Deep Learning Inference on Edge Devices," IEEE Transactions on Neural Networks, vol. 32, no. 2, 2021, pp. 488-497.
7. T. Chen, Z. Wang, and Y. Zhang, "Adaptive Precision in Floating-Point Arithmetic for Performance and Power Optimization," IEEE Design and Test, vol. 35, no. 1, 2018, pp. 89-96.
8. J. Park, D. Kim, and S. Choi, "Power-Aware Arithmetic Circuits: A Survey of Techniques and Implementations," IEEE Circuits and Systems Magazine, vol. 9, no. 2, 2020, pp. 45-

67.

1. F. Liu, M. Shafique, and A. Kumar, "Approximate Floating-Point Arithmetic for Machine Learning Accelerators," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 38, no. 4, 2019, pp. 641-654.
2. J. Han, and M. Orshansky, "Error Correction in Approximate Computing: Strategies and Hardware Implementations," IEEE Transactions on Circuits and Systems, vol. 62, no. 4, 2015, pp. 891-902.

1. L. J. Z. Li, “ Efficient Approximate Floating-Point Multiplier With Runtime

Reconfigurable Frequency and Precision,” IEEE Transactions on Circuits and Systems II: Express Briefs, pp. 3533-3537, 2024.

1. R. P. B. a. H. T. Kung, “A regular layout for parallel adders,” IEEE Transactions on Computers, Vols. C-31, p. 260–264, Mar. 1982.
2. N. R. A. W. Dehaene, "Ultra-Low-Voltage Design of Energy-Efficient Digital Circuits," Springer, 2015.
3. K. R. A. A. R. S. Venkataramani, "Approximate computing and the quest for computing efficiency," in Proceedings of the 52nd Annual Design Automation Conference (DAC), 2015.
4. C. S. K. Golda Hepzibha, "A novel implementation of high speed modified Brent Kung carry select adder," 1st International Conference on Power Electronics, Intelligent Control and Energy Systems, 2016.

**Appendex**

// Test Bench

`timescale 1 ns / 1 ps module tb\_Approximate\_Floating\_Multiplier\_32bit ();

reg clk; reg clk2; reg clk3; wire clk\_out; reg reset;

wire [31:0] A\_input; wire [31:0] B\_input; reg [31:0] Multiplication\_Result; reg A\_Sign; reg B\_Sign; reg [7:0] A\_Exponent; reg [7:0] B\_Exponent; reg [22:0] A\_Mantissa; reg [22:0] B\_Mantissa; reg [45:0] AB\_m; reg [7:0] AB\_ex; reg [53:0] Nr; reg [31:0] Check\_Data; reg [31:0] Check\_Data\_samp; wire [31:0] m1; reg [31:0] m2; reg [31:0] m3;

reg error; wire [31:0] error\_diff; parameter [1:0] K\_MODE = 2'b00; // 00 - No correciton mode, 10 - Low precision correction, 11

- High Precision correction parameter [7:0] K\_BIAS = 8'd127;

initial

begin clk <= 1'b1; reset <= 1'b0; #20 reset <= 1'b1; forever #5 clk <= ~clk; end

initial

begin clk2 <= 1'b1; forever #10 clk2 <= ~clk2; end

initial

begin clk3 <= 1'b1; forever #15 clk3 <= ~clk3; end assign A\_input = {A\_Sign,A\_Exponent,A\_Mantissa}; assign B\_input = {B\_Sign,B\_Exponent,B\_Mantissa}; assign error\_diff = (Check\_Data\_samp < Multiplication\_Result)? (Multiplication\_Result - Check\_Data\_samp) : (Check\_Data\_samp - Multiplication\_Result); always @ (posedge clk\_out or negedge reset) begin if (reset == 1'b0) begin

A\_Sign <= 1'b0;

B\_Sign <= 1'b0;

A\_Exponent <= 'b0;

B\_Exponent <= 'b0;

A\_Mantissa <= 'b0;

B\_Mantissa <= 'b0;

AB\_m <= 'b0;

AB\_ex <= 'b0; Nr <= 'b0;

Check\_Data <= 'b0; m2 <= 'b0; m3 <= 'b0; Multiplication\_Result <= 'b0; error <= 'b0; Check\_Data\_samp <= 'b0; end else begin

m2 <= m1;

m3 <= m2;

Multiplication\_Result <= m3;

A\_Exponent <= $urandom;

B\_Exponent <= $urandom;

A\_Mantissa <= $urandom;

B\_Mantissa <= $urandom;

AB\_m <= A\_Mantissa \* B\_Mantissa;

AB\_ex <= (A\_Exponent + B\_Exponent) - K\_BIAS ;

Nr <= {(AB\_ex+1'b1),AB\_m};

Check\_Data[31] <= A\_Sign^B\_Sign ;

Check\_Data[30:0] <= Nr[53:23];

Check\_Data\_samp <= Check\_Data;

if (Check\_Data\_samp == Multiplication\_Result && Multiplication\_Result != 'b0 )

begin

error <= 1'b0;

$display ("Multiplication Test Equal", Check\_Data\_samp);

end

else begin

error <= 1'b1;

$display ("Multiplication Test Approximate", Check\_Data\_samp); if (Multiplication\_Result != 'b0) begin

$display ("Error Different : %d", error\_diff ); end

end

end

end

//Approximate\_Floating\_Multiplier\_32bit FPM (

.clk1 (clk),

.clk2 (~clk2),

.clk3 (clk3),

.reset (reset),

.A (A\_input),

.B (B\_input),

.Mode (K\_MODE),

.App\_FP (m1),

.clk\_out (clk\_out)

);

Endmodule

// Approximate Floating\_Multiplier\_ 32bit

`timescale 1 ns / 1 ps module Approximate\_Floating\_Multiplier\_32bit ( input clk1,

input clk2, input clk3, input reset,

input [31:0] A, input [31:0] B, input [1:0] Mode, output wire [31:0] App\_FP, output wire clk\_out

); wire clk\_outx; wire select\_1; wire select\_2; wire select\_3; wire [2:0] SEL; wire Areg\_sign; wire [7:0] Areg\_exponent; wire [6:0] Areg\_mantissa; wire [5:0] mAh; wire Breg\_sign; wire [7:0] Breg\_exponent; wire [6:0] Breg\_mantissa; wire [5:0] mBh; wire sign\_bit;

wire [8:0] E\_sum; wire [8:0] EA\_EB; wire [6:0] Manti\_C; wire [7:0] Exponent\_out; wire [7:0] Mantissa\_out; wire [7:0] M\_sum; wire [7:0] MC\_sum; wire [2:0] Mode\_ctl; assign clk\_out = clk\_outx; assign SEL = {select\_3,select\_2,select\_1};

//Frequency\_Precision\_Controller fpcx1 (

.clk1 (clk1),

.clk2 (clk2),

.clk3 (clk3),

.reset (reset),

.Mode (Mode),

.clk\_out (clk\_outx),

.select\_1 (select\_1),

.select\_2 (select\_2),

.select\_3 (select\_3)

);

//Mode\_Register Modex1 (

.clk (clk\_outx),

.reset (reset),

.SEL ({select\_3,select\_2,select\_1}),

.Mode\_ctl (Mode\_ctl)

);

//Input\_Register Input\_inst (

.clk (clk\_outx),

.reset (reset),

.input\_a (A),

.input\_b (B),

.Areg\_sign (Areg\_sign),

.Areg\_exponent (Areg\_exponent),

.Areg\_mantissa (Areg\_mantissa),

.mAh (mAh),

.Breg\_sign (Breg\_sign),

.Breg\_exponent (Breg\_exponent),

.Breg\_mantissa (Breg\_mantissa),

.mBh (mBh)

);

//Error\_Correction\_Module ECMx1 (

.mA (mAh),

.mB (mBh),

.SEL (Mode\_ctl),

.Manti\_C (Manti\_C)

);

assign sign\_bit = Areg\_sign ^ Breg\_sign; //Brent\_Kung\_Adder\_8Bit bkax1 (

.A (Areg\_exponent),

.B (Breg\_exponent),

.Cin (1'b1),

.Sum (EA\_EB[7:0]),

.Cout (EA\_EB[8])

);

assign Exponent\_out = EA\_EB[7:0] - 8'd127; // Bias Removal

//Brent\_Kung\_Adder\_8Bit bkax2 (

.A ({1'b0,Areg\_mantissa}),

.B ({1'b0,Breg\_mantissa}),

.Cin (1'b0),

.Sum (M\_sum),

.Cout ( )

);

//Brent\_Kung\_Adder\_8Bit bkax3 (

.A (M\_sum),

.B ({1'b0,Manti\_C}),

.Cin (1'b0),

.Sum (MC\_sum),

.Cout ( )

);

assign Mantissa\_out = (EA\_EB[8] == 1'b0)? M\_sum : MC\_sum; // Mantissa processing

Exception\_Processing expx1 (

.sign\_bit (sign\_bit),

.Exponent\_out (Exponent\_out),

.Mantissa\_out (Mantissa\_out),

.App\_FP (App\_FP)

);

endmodule