**A Project Report**



**On**

**An Energy-Efficient Generic Accuracy Configurable Multiplier Based on Dual Sub-Adders with Error Recovery**

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

**Bachelor of Technology in**

**Electronics and Communications Engineering**

***by***

**MATAVALAM JAHNAVI 21F61A0469**

**DURGESAM HEMANTH KUMAR 21F61A0462**

**BURUGULA KANAKA DURGA PRASADU 21F61A0482**

**M.KARTHIK 21F61A0486**

**THANDLAM LOKESH 21F61A04A3**

***Under the esteemed guidance Of***

**Mr. RAGHUL G**

***Assistant Professor, Department of ECE***

**Department of Electronics and Communications Engineering**

# SIDDHARTH INSTITUTE OF ENGINEERING & TECHNOLOGY

**(AUTONOMOUS)**

**(Approved by AICTE & Affiliated to JNTUA, Ananthapuramu) (Accredited by NBA for Civil, EEE, ECE, MECH and CSE, New Delhi)**

**(Accredited by NAAC with ‘A+’ Grade, an ISO 9001:2008 Certified Institution) Siddharth Nagar, Narayanavanam road, Puttur-517583, A.P**

**2025**



**SIDDHARTH INSTITUTE OF ENGINEERING & TECHNOLOGY**

**(AUTONOMOUS)**

**(Approved by AICTE & Affiliated to JNTUA, Ananthapuramu) (Accredited by NBA for Civil, EEE, ECE, MECH and CSE, New Delhi)**

**(Accredited by NAAC with ‘A+’ Grade, an ISO 9001:2008 Certified Institution) Siddharth Nagar, Narayanavanam road, Puttur-517583, A.P**

**DEPARTMENT OF ELECTRONICS AND COMMUNICATIONS ENGINEERING**

*CERTIFICATE*

*This is to certify that the Project entitled “An Energy-Efficient Generic Accuracy Configurable Multiplier Based On Dual Sub-Adders With Error Recovery”that is being submitted by*

**MATAVALAM JAHNAVI 21F61A0469**

**DURGESAM HEMANTH KUMAR 21F61A0462 BURUGULA KANAKA DURGA PRASADU 21F61A0482 M.KARTHIK 21F61A0486**

**THANDLAM LOKESH 21F61A04A3**

*is in partial fulfillment of the requirements for the award of BACHELOR* OF

*TECHNOLOGY in* ELECTRONICS AND COMMUNICATIONS

ENGINEERING *to* JNTUA , ANANTHAPURAMU*. The results embodied in this Project report have not been submitted to any other University or Institute for the award of any degree.*

**Internal Guide Head of the Department**

**MR. RAGHUL G Dr.P.RATNA KAMALA, M.Tech.,Ph.D**

Assisstant Professor, Head Of the Department,

Department Of ECE, Department Of ECE,

SIETK. SIETK.

***Submitted for the project viva-voce examination held on***

**Internal Examiner External Examiner**

**Acknowledgement**

We wish to express our profound and sincere gratitude to MR.RAGHUL G **,** Assistant Professor of Electronics and Communications Engineering, **Siddharth Institute of Engineering & Technology, Puttur**, who guided us into the intricacies of this project with utmost clarity.

We would also like to extend our gratitude to **Dr. P Ratna Kamala**, Head of the Department of Electronics and Communications Engineering for her encouragement and for providing the facilities to carry out the work in a successful manner.

We are thankful to **Dr. K. Chandrasekhar Reddy**, Principal for his encouragement and support.

We wish to express our sincere thanks to **Dr. K. Indiraveni, Vice-Chairman,** and **Dr. K. Ashok Raju, Chairman** of Siddharth Group of Institutions, Puttur, for providing ample facilities to complete the project work.

We would also like to thank all the faculty and staff of the Electronics and Communications Engineering Department, for helping us to complete the project work.

Very importantly, we would like to place on record our profound indebtedness to our parents and families for their substantial moral support and encouragement given throughout our studies.

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **CHAPTER NO.** | **NAME** | **PAGE NO.** |
|  | ABSTRACT | i |
|  | LIST OF FIGURES | ii - iii |
|  | SYMBOLS & ABBREVIATIONS | iv |
| CHAPTER 1 | INTRODUCTION | 1-15 |
| CHAPTER 2 | LITERATURE SURVEY | 16-21 |
| CHAPTER 3 | PROPOSED SYSTEM | 23-26 |
| 3.1 PROPOSED SYSTEM | 23-24 |
| 3.2 BLOCK DIAGARM FLOW | 25-26 |
| CHAPTER 4 | SOFTWARE DESCRIPTION | 27-45 |
| 4.1 VERILOG | 27 |
| 4.2 DESIGN USING HDL | 29-30 |
| 4.3 SPARTAV 3 | 30-31 |
| 4.4 ARCHITECTURAL OVERVIEW | 31-32 |
| 4.5 CONFIGURABLE LOGIC BLOCK | 32-33 |
| 4.6 DOWNLOAD VIRTUALBOX | 33-40 |
| 4.7 DOWNLOAD XILINX ISE  4.8 INSTALLING XILINX IN VITRUAL BOX | 40-43  43-45 |
| CHAPTER 5 | ADVANTAGES AND APPLICATIONS   * 1. ADVANTAGES   2. APPLICATIONS | 46 |
| CHAPTER 6 | RESULTS AND DISCUSSION | 47-51 |
| CHAPTER 7 | CONCLUSION AND FUTURE SCOPE   * 1. CONCLUSION | 52 |
| 8.2 FUTURE SCORE | 52 |
|  | REFERENCES | 53-54 |
| ANNEXURE – A | SOURCE CODE | 55-59 |
| ANNEXURE – B |  | - |
| ANNEXURE – C |  | - |

# ABSTRACT

Approximate computing can decrease the design complexity with an increase in performance and power efficiency for error resilient applications. This brief deals with a new design approach for approximation of multipliers. This work proposes a generic accuracy-configurable multiplier that employs a novel dual sub-adder based approximate adder that splits a precise adder into two to significantly reduce the latency. The proposed error recovery and reduction technique effectively compensates for the catastrophic accuracy degradation incurred by the split. The proposed approximation is utilized in two variants of multipliers. Performance of the proposed multipliers is evaluated with a Xilinx 12.1.

KEY WORDS: Xilinx 12.1, sub-adder, multipliers.

# LIST OF FIGURES

|  |  |  |
| --- | --- | --- |
| **FIGURE NO.** | **FIGURE NAME** | **PAGE NO.** |
| Fig 1.1 | Low – latency approximate adder | 11 |
| Fig 1.2 | Example for approximate multiplier | 13 |
| Fig 3.2 | Block diagram flow | 25 |
| Fig 4.1 | Spartan-3 Package XC3S400-4PQ208C | 31 |
| Fig 4.2 | Spartan-3 Architecture | 32 |
| Fig 4.3 | Arrangement of Slice within the CLB | 33 |
| Fig 4.4 | Setup of Virtual Box | 34 |
| Fig 4.5 | Setup of Virtual Box Applications | 35 |
| Fig 4.6 | Custom Setup of Virtual Box | 35 |
| Fig 4.7 | Setup of Virtual Box Network Interfaces | 36 |
| Fig 4.8 | Installing The Virtual Box | 36 |
| Fig 4.9 | Installing In Processes | 37 |
| Fig 4.10 | Installing The Adapter And Serial Bus | 37 |
| Fig 4.11 | Finishing Up The Installation | 38 |
| Fig 4.12 | Setting Up Virtual Box Preferences | 38 |
| Fig 4.13 | Setting Up Extensions Of Virtual Box | 39 |
| Fig 4.14 | Selecting The Extension Pack | 39 |
| Fig 4.15 | Installing The Extension Pack | 39 |
| Fig 4.16 | Accepting The License Of Virtual Box | 40 |
| Fig 4.17 | Completing The Installation | 40 |
| Fig 4.18 | Installing The Xilinx Software | 41 |
| Fig 4.19 | Selecting The Specific Xilinx Software | 42 |
| Fig 4.20 | Move To The Download Section | 42 |
| Fig 4.21 | Downloading Latest Version Xilinx Software | 43 |
| Fig 4.22 | Setting Up Xilinx Software In Virtual Vox | 43 |
| Fig 4.23 | Allocating The Location For Installing | 44 |
| Fig 4.24 | Giving Access To The Window Folders | 44 |
| Fig 4.25 | Enabling The Virtualization Option | 45 |
| Fig 6.1 | Simulation result | 48 |

|  |  |  |
| --- | --- | --- |
| Fig 6.3 | Design Summary of Existing | 49 |
| Fig 6.4 | Design Summary of Proposed | 49 |
| Fig 6.5 | RTL Schematic | 50 |
| Fig 6.6 | Gate level Netlist | 50 |
| Fig 6.7 | Performance results | 51 |

**SYMBOLS & ABBREVIATIONS**

|  |  |
| --- | --- |
| **ACRONYM** | **ABBREVIATION** |
| **(ASICs)** | Application-specific integrated circuits |
| **(FPGAs)** | Field-programmable gate arrays |
| **(RCA)** | Ripple Carry Adder |
| **(VLSI)** | Very Large Scale Integration |

## Chapter 1 Introduction

**Introduction to Multipliers in VLSI**

In VLSI (Very Large Scale Integration) design, multipliers are crucial components in digital signal processing (DSP), image processing, communications, and computing applications. They are designed to perform the multiplication of binary or decimal numbers, which is a fundamental operation in arithmetic circuits. Given that multiplication is often more complex than addition and subtraction, multipliers must be optimized to operate quickly and efficiently, which is particularly challenging at the hardware level where speed, power consumption, and area are critical considerations. Efficient multiplier design significantly impacts the overall performance of VLSI circuits, especially in applications requiring high-speed and high-precision computations.

## Types of Multipliers in VLSI

* 1. **Array Multipliers**

Array multipliers are fundamental, straightforward designs that are built using an array of adders arranged to compute partial products sequentially. In an N×NN

\times NN×N bit array multiplier, each bit of one operand is multiplied by each bit of the other operand, producing partial products that are systematically added in an array configuration. The addition of partial products is managed by a series of adders aligned in a rectangular structure, which handles carries as they propagate through the array.

The working principle involves aligning partial products according to their binary significance, then summing them using adders. Although simple, this design can be inefficient due to the long propagation delay associated with carry propagation through multiple adder stages. Array multipliers are typically larger in area and consume more power, but their simplicity makes them advantageous for applications where design complexity is a constraint and power consumption is manageable. They are especially suitable for embedded systems where the multiplier structure can remain uncomplicated.

## Wallace Tree Multipliers

Wallace tree multipliers optimize the addition of partial products by structuring them in a hierarchical “tree” form. This architecture, named after its inventor, Chris Wallace, aims to reduce the depth of the adder stages by grouping partial products in sets of three and adding them simultaneously. Using carry-save adders (CSA), the design merges groups in parallel, significantly minimizing the number of sequential addition stages needed to obtain the final product.

The main advantage of this multiplier is its speed. By reducing the critical path delay, the Wallace tree structure achieves faster computation, which is especially valuable in high-performance applications like digital signal processing (DSP) and computer processors. However, Wallace tree multipliers are more complex to design and can require additional hardware resources due to the tree’s structural demands. This design complexity can increase both the power consumption and the layout area. Despite this, the speed gains make it a preferred choice in applications that prioritize high-speed data processing.

## Booth Multipliers

Booth multipliers use Booth’s algorithm, which is particularly efficient for signed binary numbers and allows multiplication of both positive and negative numbers. Booth’s algorithm works by encoding the multiplier operand, thereby reducing the number of partial products generated. The algorithm identifies consecutive 1’s in the multiplier and minimizes them by encoding, which shortens the calculation and reduces power consumption.

In operation, Booth’s algorithm performs shifts and additions based on specific rules, reducing the required number of operations for certain bit patterns, particularly for large bit-widths. However, the complexity of the logic involved in recoding and handling signed binary numbers can slightly increase power consumption and area compared to simpler methods. Booth multipliers are commonly used in applications needing efficient handling of signed numbers, such as in DSP and multimedia processing, where the ability to handle negative values is essential.

## Vedic Multipliers

Vedic multipliers are based on principles derived from ancient Vedic mathematics, specifically the “Urdhva Tiryakbhyam” or “vertically and crosswise” technique. This method breaks down numbers into smaller parts and performs

multiplication on these parts simultaneously. The vertical and crosswise approach minimizes the number of intermediate steps by generating partial products and summing them almost in parallel.

The Vedic multiplier’s working principle allows for faster computations as it reduces the dependency on sequential addition stages. This architecture’s high- speed and low-power characteristics make it ideal for resource-constrained applications, particularly in embedded systems and mobile devices. The reduced complexity in partial product generation also leads to a smaller footprint, which conserves area and power. Vedic multipliers are popular in fields where both speed and energy efficiency are paramount, such as in real-time systems, robotics, and mobile applications.

## Carry-Save Multipliers

Carry-save multipliers are designed using carry-save adders (CSA), which streamline the process of adding partial products by deferring carry propagation until the final addition step. Unlike other multipliers where each adder must propagate its carry to the next stage immediately, carry-save adders accumulate carries separately. At the final stage, a traditional adder resolves these deferred carries, producing the final result.

The working mechanism of carry-save multipliers avoids delays associated with carry propagation at intermediate stages, reducing overall delay and enhancing performance. Carry-save multipliers are, therefore, suited for applications requiring high-speed computations, such as in DSP and central processing units (CPUs). They are particularly beneficial in scenarios where multiple products must be summed quickly, as in matrix multiplications and filtering operations in DSP. This architecture's drawback is the increased number of adders, which can lead to a larger area and higher power usage.

## Approximate Multipliers

Approximate multipliers are innovative designs aimed at balancing resource usage and computational efficiency by intentionally reducing accuracy. In applications such as machine learning, computer vision, and audio processing, exact multiplication results are not always essential. Approximate multipliers simplify the logic and minimize resource requirements, achieving lower power

consumption, reduced latency, and decreased area usage by sacrificing some precision.

These multipliers work by truncating or simplifying parts of the calculation, such as by rounding partial products or using fewer bits for multiplication. This results in a smaller, faster circuit that consumes less power, though the trade-off is a slight reduction in accuracy. Approximate multipliers are advantageous in energy-efficient designs and scenarios where computational throughput is more critical than perfect precision, such as in deep learning and signal processing accelerators. This type of multiplier is particularly useful for applications in which power efficiency is a priority and minor accuracy losses are tolerable.

Low-power VLSI design focuses on reducing power consumption in integrated circuits (ICs), which is essential in modern electronics for improving battery life in mobile devices, reducing heat in high-performance systems, and achieving overall energy efficiency. With the increase in IC complexity and the demand for compact, portable, and high-performance electronics, managing power consumption has become a priority in VLSI design. Power reduction techniques in VLSI are classified into multiple categories, including logic-level and circuit-level modifications, each addressing specific aspects of the design to minimize power usage while maintaining performance.

## Power Consumption in VLSI and Its Types

Power consumption in VLSI circuits can be broadly divided into dynamic and static power. Dynamic power is consumed during switching activities, i.e., when transistors transition between logic states, and is a function of factors like clock frequency, switching activity, and capacitance. Static power, or leakage power, occurs even when the circuit is idle due to sub-threshold leakage currents in transistors. As technology scales down, leakage power becomes a significant component of overall power consumption.

## Techniques for Low-Power VLSI Design

Several techniques have been developed to address power consumption in VLSI circuits, targeting both dynamic and static power. These methods include:

## Logic-Level Modifications

1. **Circuit-Level Modifications**

Each of these techniques involves specific strategies for achieving power efficiency.

## Logic-Level Modifications

Logic-level modifications focus on optimizing the logic design and data flow within the circuit to reduce unnecessary switching activities, thereby saving dynamic power. These techniques aim to minimize the amount of activity in the logic gates by optimizing the circuit's logic structure without affecting functionality.

* **Gate Sizing**: Adjusting the size of transistors in logic gates to the minimum size required for the performance. Smaller gates consume less power, though they may slow down the circuit in some cases.
* **Clock Gating**: One of the most effective ways to reduce power, clock gating disables the clock signal to certain parts of the circuit when those parts are not in use, effectively reducing switching power. For example, registers and counters may only be active when new data is available, so the clock signal can be gated off to those modules during inactive periods.
* **Operand Isolation**: In certain designs, operand isolation uses control logic to cut off data inputs to certain logic blocks when they are not required, preventing unnecessary switching and reducing power usage.
* **Logic Minimization**: Logic minimization algorithms, such as Karnaugh maps or Boolean algebra techniques, help in reducing the number of gates or operations needed to implement a function. By simplifying the logic, fewer gates are needed, leading to lower power consumption.

Logic-level techniques are particularly advantageous because they can achieve substantial power reductions with minimal impact on circuit area and speed.

## Circuit-Level Modifications

Circuit-level modifications focus on optimizing the transistor circuits and internal elements of the ICs to further minimize power consumption. This level involves changes to the underlying transistor circuits used to implement logic gates and other components.

* **Voltage Scaling**: Reducing the supply voltage (Vdd) is one of the most effective ways to reduce power consumption, as dynamic power is proportional to Vdd2V\_{dd}^2Vdd2. However, lower voltages can also reduce speed, so

designers must balance power savings with performance needs. For instance, dynamic voltage and frequency scaling (DVFS) adjusts voltage and frequency based on workload requirements, lowering power usage during low-demand periods and increasing performance when needed.

* **Sub-Threshold Logic**: In sub-threshold logic, the supply voltage is reduced below the threshold voltage of transistors, drastically cutting power consumption at the expense of performance. While this approach may lead to increased delays, it is valuable for ultra-low-power applications such as IoT devices and sensors where speed requirements are minimal.
* **Multi-Threshold CMOS (MTCMOS)**: This technique uses transistors with different threshold voltages within the same design. High-threshold transistors reduce leakage currents for non-critical paths, while low-threshold transistors provide higher speeds for critical paths. MTCMOS is particularly useful in reducing leakage power in idle parts of a circuit.
* **Power Gating**: Power gating disconnects power from parts of the circuit when they are inactive, thereby reducing leakage power. By turning off entire sections of a circuit when they are not required, significant power savings are achieved, though extra circuitry for control is needed. Power gating is ideal for components that remain inactive for extended periods.
* **Body Biasing**: In body biasing, the body (substrate) of a transistor is biased to increase or decrease its threshold voltage. By dynamically adjusting the threshold voltage, body biasing helps to reduce leakage currents in idle states and improve performance during active states. Forward body biasing can reduce the threshold for active regions, while reverse body biasing can increase it for inactive regions.

## Importance and Challenges of Low-Power VLSI Design

Low-power VLSI design is essential in modern ICs due to the growing demand for portable electronics, high-performance computing, and energy- efficient solutions. However, achieving low-power design is challenging as power reduction techniques can impact other design parameters such as speed, area, and complexity. For instance, voltage scaling can slow down circuits, while logic-level and circuit-level modifications may increase design complexity and development time.

In summary, low-power VLSI design is a critical field that employs various techniques at the logic and circuit levels to optimize power usage. By combining these methods thoughtfully, designers can create energy-efficient ICs that meet the demands of modern applications without compromising functionality or performance.

## Key Considerations in Multiplier Design

1. Speed

Speed is often the primary concern in multiplier design, especially for high- performance processors and DSP applications. Designs such as Wallace tree and carry-save multipliers focus on reducing propagation delay and the number of sequential additions required.

1. Power Consumption

In battery-operated devices, low power consumption is essential. Vedic multipliers and approximate multipliers are beneficial in these cases, as they are designed to minimize power usage, often by reducing the complexity of the circuit.

1. Area

The area of the multiplier circuit affects the chip size and cost. Some multipliers, like array multipliers, may require more area but are simpler to implement, while Wallace tree and carry-save multipliers can be optimized for high-speed and area- efficient applications.

1. Accuracy

Accuracy is crucial in applications where precise calculations are necessary, but in fields like machine learning, approximate multipliers offer acceptable trade-offs between accuracy and resource efficiency. For signed and unsigned number operations, Booth multipliers are often preferred due to their handling of negative numbers.

## Applications of Multipliers in VLSI

Multipliers are integral to a wide range of applications, particularly in DSP and machine learning accelerators where repeated multiplication is a core operation. In image processing, multipliers enable fast filtering, transformation, and scaling operations. In cryptography, they assist in encryption algorithms that rely on complex arithmetic. Additionally, multipliers are foundational in neural

network hardware accelerators, where efficient matrix multiplications are necessary for high-throughput computations.

This in-depth exploration of multipliers in VLSI highlights their diversity, with each type catering to specific requirements in terms of speed, power, area, and accuracy. The optimal choice depends on the application’s demands and the design goals for performance and efficiency.

# APPROXIMATION METHODOLOGIES

Based on the literature survey, it is known that several methods have been adapted in designing Approximate Circuits. Approximations of computation can be performed at architecture, software and/or circuit level . Circuit- approximations involve the construction of approximate models of logic circuits which are utilizing techniques such as Systematic Logic Synthesis of approximate Circuits (SALSA). Another common approach to circuit- level approximations is the design of arithmetic data paths using approximate adders and/or multipliers. Approximations of the degree of complexity may be rendered in sophisticated architectures, such as Artificial Neural Networks (ANNs), by recognizing essential neurons. Software may also be Approximated using methods such as Computer. The following is a description of the Approximate programming methods through hardware and software stacks.

## Outline of the Approximate Computing Methods Software

Selective Approximation -Using manually annotated or automatically interpreted code to adaptively bypass error-resilient portions of the application as per user-defined consistency specifications.

Timing Relaxation - Synchronization between concurrent systems or hand- shaking between separate code segments should be relaxed in order to boost energy efficiency and performance.

Domain Specific Approximation - Using domain / application-specific information by classifying data into categories so that sensitive / complex data is processed using correct computation and insensitive data is processed by using approximate modules.

Functional Approximation - Energy-hungry Approximate code fragments are supplemented by approximate alternatives that are either performed using correct or different approximate hardware modules.

## Architectural

Selective Approximation – Utilizes specialized approximate hardware components to perform specific instructions / code segments.

Domain Specific Approximation - Integrates Domain / Application relevant expertise to develop proficient approximate architectures.

Functional Approximation - Simplifies the design sophistication of the hardware by utilizing the data type of essential paths or the general implementation of dynamic modules by utilizing their approximate counterpart.

Data Approximation - Relaxed reliability / reliability memory architectures (specifically by easing error correction limits or intelligently handling read / write operations at hardware error-resilient data level) contribute to major developments in resource efficiency and performance.

## Hardware Circuit

Timing Relaxation- Reduction in input voltage to increase overall power and energy efficiency.

Functional Approximation - Deliberate decrease in the amount of hardware element transistors / gates to be replaced by less complicated equivalent approximate models.

Data Approximation - Utilizes low consistent memory modules which are designed by using lesser amount of transistors.

# COMPONENT-LEVEL APPROXIMATIONS FOR ARITHMETIC MODULES

Multipliers and adders are the basic building blocks in all arithmetic modules including general purpose processors and adapted on-chip accelerators. Due to this fact that mostly applications which are agreeable to Approximate Computing are computationally demanding requires huge number of arithmetic functions. The design of approximate multipliers and adders have gained great attention. An

overview of approximate adders and multipliers are given in brief in the following subsections.

# APPROXIMATE ADDERS

Adders are one among the most commonly employed operators in arithmetic systems. Approximate adders exploit practical simplifications to provide efficiency and/or area / power advantages. There are primarily two complementary methods for the creation of estimated adders: (1) by minimizing the logic / circuitry of the basic blocks of the adders, i.e. by approximating the complexity and thereby simplifying the logic of the complete adders; and (2) by approximating the logic of the carry chain truncation. All strategies are discussed extensively in the following sub-sections.

## Approximate Adders with Low Power

Ripple Carry Adder (RCA) is the most common and basic adder design consisting of cascaded full-adder modules. The most influential aspect of RCA is its low power usage relative to all other accurate adder architectures. Applications with tight resource restrictions need much easier and lesser resource consuming modules. Using the usable low-power full-adder modules, low-power RCAs may be built by replacing accurate full-adders with their approximate counterpart. Full- adders close to LSB positions have fewer importance relative to the bits close to MSB, thus, in most instances, LSBs are approximated using various types of Approximate-full-adders.

## Low-Latency Approximate Adders

In order to fulfill the high output performance of error-tolerance systems, the truncation of its critical pathways may be abused. In the case of adders, it is also usually accomplished by utilizing several intersecting smaller sub-adder units that run in parallel to produce output with comparatively lower latency.

Sub Adder 1



Co

Output[R+P:1] Sub Adder 2

Co

A[N:1]

.

output[2R+P:R+P+1] output[N+1:1]

B[N:1]

Sub Adder K

Co

Output[kR+P:(k-1)R+P+1]

### Figure 1.1 Low – latency approximate adder

Occasionally, such adders are often equipped with error identification and correction logic to minimize the error likelihood by dynamically modifying the carry getting disseminated to subsequent levels.

Thorough and time-efficient space exploration of usable low-latency adders require a coherent model that is highly parameterizable and capable of accommodating most (but not all) forms of low-latency adders. To this end, we have designed the GeAr (Shafique *et al.* 2015) adder structure, that is highly customizable and able to model so many of the state-of – the-art high-efficient adders. For adding two N-bit operands, the GeAr adder uses k number of L-bit sub- adders that work in parallel to produce an Approximate output with high rate of speed, where L < = N is used. Figure 1.1 displays the schematic form of the adder layout. Let R be the number of resulting bits that leads to the final output and P be the number of carry-forecast bits used to estimate the carry in all sub-adder. Every sub-adder generates R-bits of output except for the first sub-adder that generates L-bits of output where L = R+P. The number of necessary sub-adders k for the given N, R, and P can be determined by k = (N-P)/R. In fact, for a combined effect of N, R and P is also be true for the GeAr model, the combination would lead to a positive integer 'k.' The configuration of the

low-latency adders (shown in Figure 1.1) indicate that an error is produced when the P-bits of each sub-adder spreads the carry and the carry is produced by the preceding sub-adder.

# APPROXIMATE MULTIPLIERS

Efficient hardware multiplication is usually done in three stages:

(1) generating partial products; (2) intermediate partial product accumulation; and

(3) accumulation of the final results by fast adder. Approximations may be used at each point to increase the power, area, efficiency or output performance of the multipliers. Next approach to design intensive multipliers would be to utilize the smaller multipliers as basic components to build larger multipliers. Figure 1.2 shows an illustrative example of such techniques.



|  |  |
| --- | --- |
|  |  |
|  |  |



W/2\*W/2 Multiplier block

W/2



W



### Figure 1.2 Example for approximate multiplier



The main building blocks which were used in the preceding methods are 2x2 multipliers, compressors, half and full adders. A variety of Approximate architectures for 2x2 multipliers (Kulkarni *et al.* 2011), compressors (Momeni *et al.* 2015), and full-adders (Gupta *et al.* 2011) have been suggested in the existing methods, that can be used to design Approximate multipliers. In this sense, we have performed a detailed architectural-space study of Approximate multipliers (Rehman *et al.* 2016) using: (1) different kinds of elementary multiplier modules, (2) different forms of elementary adder subsystems for the accumulation

of partial products, and (3) choice of bits for approximation in such a large-bit multiplier structure.

# PROBABILISTIC ERROR ANALYSIS

Throughout the traditional digital architecture, the output metrics are used to define the circuit are: 1) power consumption, 2) latency, 3) critical path delay and

4) silicon area. Likewise, tradeoffs for the attainment of specific operational requirements are often within the same criteria. Nevertheless, in the context of Approximate Computing, there is an additional criterion of efficiency, that is, computational accuracy. Approximate circuits are then expected to be eligible in the aspects of the error statistics.

The goal of the probabilistic error analysis is to logically test the precision of the approximate circuit functional model and the probability spectrum of inputs.

## Scope

The scope of this work lies in advancing the field of approximate computing for applications where error resilience is acceptable and high computational efficiency is critical. Approximate multipliers are highly relevant for domains such as image and video processing, machine learning, signal processing, and low- power IoT devices—applications where minor inaccuracies do not significantly impact overall performance but where power, area, and speed constraints are crucial. By introducing an accuracy-configurable multiplier with a novel dual sub- adder design, this work aims to address the limitations of traditional multipliers in terms of latency, power consumption, and hardware complexity.

The proposed system expands the boundaries of conventional multiplier design by offering adaptable accuracy settings that can be tuned based on the application's specific needs, paving the way for customizable hardware in error- tolerant systems. The integration of an effective error recovery technique enhances the usability of approximate multipliers in critical systems by minimizing catastrophic accuracy losses, ensuring the multiplier’s applicability in a broader range of scenarios.

Furthermore, the system's validation on Xilinx 12.1 demonstrates its potential for real-world hardware implementation, ensuring that future work can build upon this design with confidence in its scalability and feasibility. This scope thus opens doors for future research into more sophisticated error-reduction methods and optimization of approximate multipliers for application-specific integrated circuits (ASICs) and field-programmable gate arrays (FPGAs), creating significant opportunities for advancements in energy-efficient, high-speed computing systems.

## Objective

The primary objective of this work is to develop an accuracy-configurable approximate multiplier that significantly enhances performance and power efficiency while maintaining acceptable accuracy for error-resilient applications. The aim is to design a multiplier architecture that balances latency, power consumption, and computational accuracy**,** enabling its use in domains where high- speed and low-power operations are critical, such as image processing, machine learning, and signal processing**.**

A key goal is to introduce a novel dual sub-adder based approximate adder, which splits the traditional precise adder into two sub-adders, thereby reducing delay and hardware complexity. Additionally, the objective includes implementing a robust error recovery and reduction technique to effectively manage the accuracy degradation caused by this approximation approach.

Furthermore, the design aims to provide flexibility by offering two variants of the multiplier, each optimized for different levels of accuracy and performance to cater to diverse application requirements. Through extensive evaluation on Xilinx 12.1, this work seeks to validate the proposed multiplier's effectiveness in terms of speed, power efficiency, and adaptability, ultimately demonstrating its potential for deployment in real-world, error-tolerant systems.

## Chapter 2 LITERATURE SURVEY

N. Amirafshar, et al propose a methodology for designing approximate N- bit array multipliers based on carry disregarding. Evaluate and analyze the proposed multipliers both experimentally and theoretically. The proposed 8-bit multipliers, compared to the exact multiplier, reduce the critical path delay, power consumption, and area by 29%, 29%, and 30%, on average. Compared to the existing approximate array architectures in the literature, they have improved 14.3%, 22.8%, and 26.4%, respectively. Compared to the exact 16-bit multiplier, the proposed 16-bit multipliers have reduced the delay, power consumption, and area by 35%, 24%, and 23% on average. In an image processing application, we have also demonstrated the applicability of a wide range of proposed multipliers, which have Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index Measure (SSIM) over 30 dB and 94%, respectively.

Y. Zhang et al., a hybrid bit-splitting generator (HBSG) is proposed to efficiently produce parallel bitstreams in a single clock cycle to reduce delay. The HBSG uniformly splits binary numbers into R segments, each of which is encoded in parallel by using hardwired connections according to the weight of each bit. A binary-interfaced parallel stochastic multiplier (BipSMul) using the HBSG is then proposed to accelerate the multiplication in SC. Experimental results show that the BipSMul is more energy efficient than the state-of-the-art parallel and serial stochastic designs, as well as their binary and Booth counterparts, in delay, power- delay product (PDP), and area-delay product (ADP).

S. Venkatachalam et al deals with a new design approach for approximation of multipliers. The partial products of the multiplier are altered to introduce varying probability terms. Logic complexity of approximation is varied for the accumulation of altered partial products based on their probability. The proposed approximation is utilized in two variants of 16-bit multipliers. Synthesis results reveal that two proposed multipliers achieve power savings of 72% and 38%, respectively, compared to an exact multiplier. They have better precision when compared to existing approximate multipliers. Mean relative error figures are as low as 7.6% and 0.02% for the proposed approximate multipliers, which are better than the previous works. Performance of the proposed multipliers is evaluated with

an image processing application, where one of the proposed models achieves the highest peak signal to noise ratio.

D. Esposito et al proposes novel approximate compressors and an algorithm to exploit them for the design of efficient approximate multipliers. By using the proposed approach, synthesized approximate multipliers for several operand lengths using a 40-nm library. Comparison with previously presented approximated multipliers shows that the proposed circuits provide better power or speed for a target precision. Applications to image filtering and to adaptive least mean squares filtering are also presented in the paper.

F. Sabetzadeh, et al presents an ultra-efficient approximate multiplier with error compensation capability. The proposed multiplier considers the least significant half of the product a constant compensation term. The other half is calculated precisely to provide an ultra-efficient hardware-accuracy trade-off. Furthermore, a low-complexity but effective error compensation module (ECM) is presented, significantly improving accuracy. The proposed multiplier is simulated using HSPICE with 7nm tri-gate FinFET technology. The proposed design significantly improves the energy-delay product, on average, by 77% and 54% compared to the exact and existing approximate designs. Moreover, the proposed multiplier’s accuracy and effectiveness in neural networks and image multiplication are evaluated using MATLAB simulations. The results indicate that the proposed multiplier offers high accuracy comparable to the exact multiplier in NNs and provides an average PSNR of more than 51dB in image multiplication. Accordingly, it can be an effective alternative for exact multipliers in practical error-resilient applications.

M. Zhang, et al proposes a novel 4 gate 4–2 approximate compressor which is complementary with other compressors from earlier work and constructs a hybrid multiplier based on the compressors, a constant approximation, and error correction AND gate. According to the simulation results, the proposed hybrid approximate multiplier has excellent accuracy and electrical performance tradeoff and reduces the power-delay-area product (PDAP) by 66% with an MRED of 2.5% when compared to the exact multiplier.

Z. Aizaz et al propose a truncation based Booth multiplier with a compensation circuit generated by selective modifications in k-map to circumvent

the carry appearing from the truncated part. By judicious mapping, hardware pruning and output error reduction is achieved simultaneously. In the quest of power and accuracy trade-off, Truncated and Approximate Carry based Booth Multipliers (TACBM) are proposed with a range of designs based on truncation factor w . When compared with the state-of-the-art multipliers, TACBM outperforms in terms of accuracy and Area-Power savings. TACBM( w=10 ) provides with 0.02% MRED and 23% reduction in Area-Power product compared to exact Booth multiplier. The multipliers are evaluated using image blending and Multilayer perceptron (MLP) neural network and a high value of accuracy (95.63%) for MLP is achieved.

G. Park et al present a novel design methodology of cost-effective approximate radix-4 Booth multipliers, which can significantly reduce the power consumption of error-resilient signal processing tasks. In contrast that the prior studies only focus on the approximation of either the partial product generation with encoders or the partial product reductions with compressors, the proposed method considers two major processing steps jointly by forcing the generated error directions to be opposite to each other. As the internal errors are naturally balanced to have zero mean, as a result, the proposed approximate Booth multiplier can minimize the required processing energy under the same number of approximate bits compared to the previous designs. Simulation results on FIR filtering and image classification applications reveal that the proposed approximate Booth multiplier shows the most attractive energy-performance trade-offs, achieving 28% and 34% of energy reduction compared to the exact Booth multiplier, respectively, with negligible accuracy loss.

H. Waris, et al addressed by approximating the odd multiples of radix-8 to their nearest power of two such that the errors complement each other. In the pursuit of an accuracy-energy trade-off, hybrid low radix (HLR) based two approximate Booth multipliers (HLR-BM1 and HLR-BM2) are designed. HLR-BM2, compared to the previous best error-optimized design (ABM1), achieved a reduced energy of 22% with a comparable MRED. Moreover, HLR-BM2 achieves an improvement of 75% in MRED and 11% in energy consumption compared to the previously best energy-optimized design (RAD64). As a case study, performance for image

transformation is evaluated. A high peak signal-to-noise ratio (PSNR, close to 50dB) is obtained for the proposed multiplier designs.

W. Liu, et al designs of approximate adders and multipliers based on ML are proposed; the proposed multipliers utilize approximate compressors and a reduction circuitry with so-called complement bits. An influence factor is defined and analyzed to assess the importance of different complement bits depending on the size of the multiplier; a scheme for selection of the complement bits is also presented. The proposed designs are evaluated using hardware metrics (such delay and gate complexity) as well as error metrics. Compared with other ML-based designs found in the technical literature, the proposed designs are found to offer superior performance. Case studies of error-resilient applications are also presented to show the validity of the proposed designs.

W. Liu et al., the design of approximate redundant binary (RB) multipliers is studied. Two approximate Booth encoders and two RB 4:2 compressors based on RB (full and half) adders are proposed for the RB multipliers. The approximate design of the RB-Normal Binary (NB) converter in the RB multiplier is also studied by considering the error characteristics of both the approximate Booth encoders and the RB compressors. Both approximate and exact regular partial product arrays are used in the approximate RB multipliers to meet different accuracy requirements. Error analysis and hardware simulation results are provided. The proposed approximate RB multipliers are compared with previous approximate Booth multipliers; the results show that the approximate RB multipliers are better than approximate NB Booth multipliers especially when the word size is large. Case studies of error-resilient applications are also presented to show the validity of the proposed designs.

W. Liu, et al approximate Booth multipliers are designed based on approximate radix-4 modified Booth encoding (MBE) algorithms and a regular partial product array that employs an approximate Wallace tree. Two approximate Booth encoders are proposed and analyzed for error-tolerant computing. The error characteristics are analyzed with respect to the so-called approximation factor that is related to the inexact bit width of the Booth multipliers. Simulation results at 45 nm feature size in CMOS for delay, area and power consumption are also provided. The results show that the proposed 16-bit approximate radix-4 Booth multipliers

with approximate factors of 12 and 14 are more accurate than existing approximate Booth multipliers with moderate power consumption. The proposed R4ABM2 multiplier with an approximation factor of 14 is the most efficient design when considering both power-delay product and the error metric NMED. Case studies for image processing show the validity of the proposed approximate radix-4 Booth multipliers.

V. Leon et al propose an approximate hybrid high radix encoding for generating the partial products in signed multiplications that encodes the most significant bits with the accurate radix-4 encoding and the least significant bits with an approximate higher radix encoding. The approximations are performed by rounding the high radix values to their nearest power of two. The proposed technique can be configured to achieve the desired energy-accuracy tradeoffs. Compared with the accurate radix-4 multiplier, the proposed multipliers deliver up to 56% energy and 55% area savings, when operating at the same frequency, while the imposed error is bounded by a Gaussian distribution with near-zero average. Moreover, the proposed multipliers are compared with state-of-the-art inexact multipliers, outperforming them by up to 40% in energy consumption, for similar error values. Finally, we demonstrate the scalability of our technique.

M. Masadeh, et al propose a lightweight and efficient machine-learning- based approach to build an input-aware design selector, i.e., quality controller, to adapt the approximate design in order to meet the target output quality (TOQ). For illustration purposes, we use a library of 8-bit and 16-bit energy-efficient approximate array multipliers with 20 different settings, which are commonly used in image and audio processing applications. The simulation results, based on two sets of images, including an 8 Scene Categories Dataset, which is a benchmark of images data set, demonstrate the effectiveness of the lightweight selector where the proposed tunable design achieves a significant reduction in quality loss with relatively low overhead.

A. Sadeghi, et al 8-transistor and 14-transistor 4:2 compressors are proposed. Both compressors exploit CMOS technology and a constant and conditional approximation of selected inputs, exhibiting fewer negative errors. As a result, a resource-expensive error recovery module is eliminated, yielding superior performance as compared with prior art. The 14-transistor architecture

yields a lower error rate compared to the 8-transistor architecture, trading off lower area for higher accuracy. The compressor-tailored circuit architecture is also proposed and evaluated using image multiplication. The proposed multiplier exhibits 50% area savings and 93% lower power-delay-product compared to the exact multiplier, as well as higher accuracy, and 38% PDP enhancement compared with the state-of-the-art.

W. Pan et al propose a novel 8-bit signed multiplier based on the Pass Transistor Logic (PTL) that outperforms existing designs. The post-layout simulation results show that the proposed design reduces area, delay, and power by 13.45%, 9.72%, and 15.19%, respectively, compared to the multiplier synthesized using Synopsys Design Compiler (DC). Compared to other 8-bit multipliers proposed in references, our design also shows at least a 27.66% improvement in power-delay product (PDP). Additionally, the proposed circuits exhibit superior performance at different operating voltages and process corners.

M. H. Haider, et al introduces a new approximation scheme for Booth multipliers that can operate with negligible error rates using only N/4 Booth decoders, instead of the traditional N/2 Booth decoders. The proposed 16-bit BD16.4 approximate Booth multiplier reduces the Normalized Mean Error Deviation (NMED) by 96.5% and the Power-Area-Product (PAP) by 69.6%, when compared to a state-of-the-art approximate logarithmic multiplier. Additionally, the proposed BD16.4 approximate multiplier reduces the NMED by 94.4% and PAP by 74.8%, when compared to a state-of-the-art higher-radix approximate Booth multiplier. The proposed 8-bit approximate Booth multipliers reduce the NMED by up to 74% and PAP by up to 5% when compared to the existing state-of-the-art approximate logarithmic multipliers. We validated the results derived in this paper through a neural network inference experiment, where the proposed approximate multipliers showed a negligible drop in inference accuracy compared to the exact Booth multipliers and the state-of-the-art approximate logarithmic multipliers (ALM). The proposed approximate multipliers achieved a Power-Delay-Product reduction of 63% (vs. exact) and 21.22% (vs. ALM) in 16-bit experiments and a reduction of 67% (vs. exact) and 8.75% (vs. ALM) in 8-bit experiments.

J. Vafaei et al propose a high precision redundancy multiplier (HPR-Mul) that relies on the principles of approximate computing to achieve higher energy

efficiency and lower area, as well as resolve the aforementioned challenges of the typical TMR schemes, while retaining the required level of reliability. The HPR- Mul is composed of full precision (FP) and two reduced precision (RP) multipliers, along with a simple voter to determine the output. Unlike the state-of-the-art RP redundancy multipliers (RPR-Muls) that require a complex voter, the voter of the proposed HPR-Mul is designed based on mathematical formulas resulting in a simpler structure. Furthermore, we use the intermediate signals of the FP multiplier as the inputs of the RP multipliers, which significantly enhance the accuracy of the HPR-Mul. The efficiency of the proposed HPR-Mul is evaluated in a 15-nm FinFET technology, where the results show up to 70% and 69% lower power consumption and area, respectively, compared to the typical TMR-based multipliers. Also, the HPR-Mul outperforms the state-of-the-art RPR-Mul by achieving up to 84% higher soft error tolerance.

## Chapter 3 Proposed system

The proposed system introduces an innovative approach to approximate computing by designing an accuracy-configurable multiplier, aimed at enhancing performance and power efficiency for applications that can tolerate some level of error. In this system, a novel dual sub-adder based approximate adder is utilized to reduce latency by splitting a precise adder into two smaller sub-adders. This design significantly decreases the delay and area requirements, which are often bottlenecks in multiplier design. The dual sub-adder approach thus enables faster computation while maintaining a balance between accuracy and efficiency.

To address the potential accuracy degradation caused by splitting the adder, the proposed system incorporates an error recovery and reduction technique. This technique actively compensates for the errors introduced by the approximation, effectively reducing catastrophic accuracy losses. The error recovery method ensures that the multiplier can adapt its accuracy based on specific application requirements, making it suitable for applications where slight inaccuracies are acceptable but catastrophic errors are not.

Two distinct variants of the multiplier are designed within this framework. These variants employ the proposed approximate adder to achieve high power efficiency and low latency while being configurable for different accuracy levels. The first variant prioritizes speed and power savings, ideal for real-time applications that can tolerate more significant approximations. The second variant, meanwhile, offers a more balanced approach, providing a higher degree of accuracy while still benefiting from reduced latency and energy consumption.

The performance of the proposed multipliers is thoroughly evaluated using Xilinx 12.1, a simulation tool known for accurate hardware synthesis and power analysis. Through these evaluations, the proposed system demonstrates significant improvements in design complexity, power efficiency, and operational speed compared to conventional precise multipliers, making it an excellent solution for error-resilient applications like image processing, machine learning, and real-time data analysis.

The proposed system consists of several core modules that work together to achieve high performance, power efficiency, and configurable accuracy in multiplier design. Each module plays a critical role in the system, focusing on different aspects such as approximation, error management, and evaluation.

### Approximate Adder Module

At the heart of the multiplier’s design is the **approximate adder module**. This module employs a novel dual sub-adder architecture, where a conventional precise adder is split into two sub-adders to reduce overall latency and simplify hardware. By partitioning the adder, this module decreases both delay and power consumption, making it suitable for real-time, energy-efficient applications.

However, this approximation introduces minor errors, which the system manages through further error reduction techniques.

### Error Recovery and Reduction Module

This module is crucial for maintaining acceptable accuracy levels within the multiplier. It compensates for the potential accuracy losses caused by the approximate adder split, mitigating catastrophic errors that could impact system performance. By dynamically managing the magnitude of approximation errors, this module allows the multiplier to be used in applications that require configurable accuracy, thereby broadening its usability across different error- tolerant scenarios.

### Accuracy Configuration Module

The **accuracy configuration module** enables the system to adapt its accuracy level according to specific application needs. This module provides flexibility, allowing the user to toggle between different levels of approximation in two distinct variants of the multiplier. This adaptability ensures that the multiplier can meet the varying requirements of different applications, such as prioritizing either speed and power efficiency or a higher degree of accuracy.

### Multiplier Variants Module

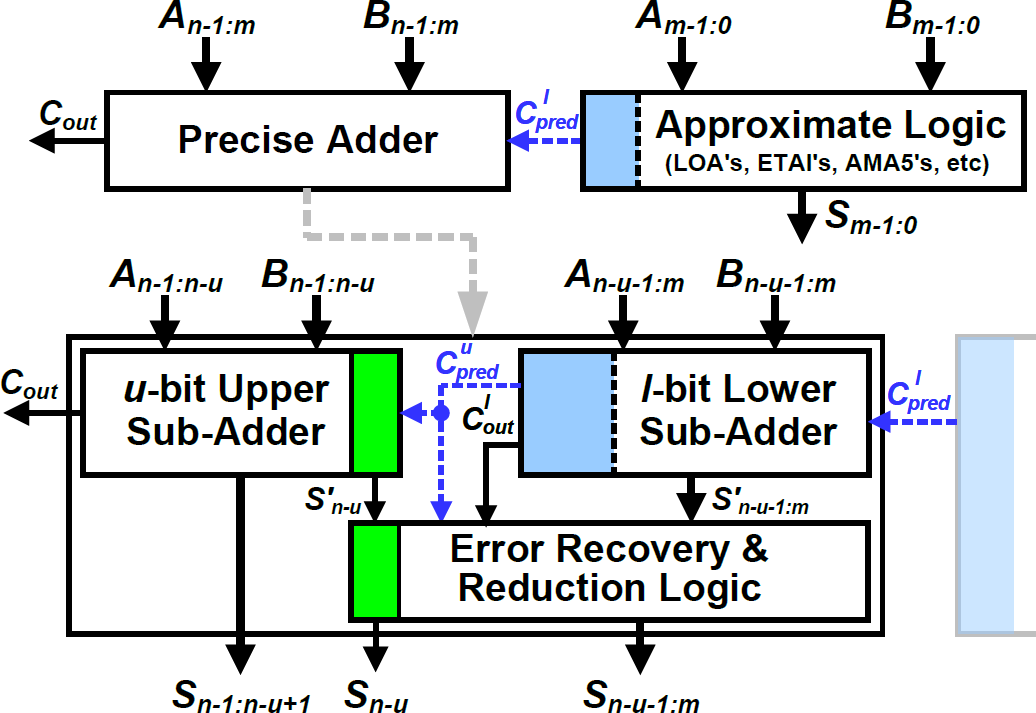
This module is responsible for implementing two separate variants of the multiplier, each optimized for a specific accuracy-performance balance. The first variant focuses on high-speed, low-power operation by maximizing the level of approximation, making it ideal for applications that can tolerate greater inaccuracies. The second variant emphasizes balanced performance, offering an

improved accuracy profile with moderate power efficiency, which suits applications requiring higher precision.

### Evaluation and Testing Module

To validate the design, the **evaluation and testing module** utilizes Xilinx 12.1 for comprehensive performance analysis. This module assesses the system in terms of latency, power consumption, and accuracy, ensuring that the design meets the targeted specifications. By testing each variant under various operational conditions, this module provides insights into the effectiveness of the proposed approximation techniques and confirms the multiplier’s potential for real-world deployment.

Each module is integral to the system’s overall functionality, enabling the approximate multiplier to achieve an optimal balance between power efficiency, speed, and configurability for error-resilient applications.



. Figure shows the block diagram of the proposed nbit approximate adder, termed dual sub-adder based low latency adder (DSLLA). While an approximate adder based on the split architecture includes one precise adder and one approximate logic, as shown in the upper of Figure , the proposed adder splits the precise adder into two smaller sub-adders to achieve low latency by cutting a long carry chain.

Thus, it forms two sub-adders, called an u-bit upper sub-adder (USA) and an l-bit lower sub-adder (LSA), and one m-bit approximate logic. Note that these sub- adders can be implemented in any kind of precise adders, such as RCA. The USA and LSA accurately add higher-order inputs A/Bn−1:n−u with a carry Cu pred predicted from the LSA and lower-order inputs A/Bn−u−1:m with a carry Cl pred predicted from the approximate logic,respectively, to produce the corresponding partial summations simultaneously. These outcomes, together with Cu pred and the LSA’s carry out Cl out, are fed into the error recovery and reduction logic to recover an error if possible, or reduce an error distance. In this

architecture, the approximate logic can exploit any existing approximation schemes, such as LOA’s OR, AMA5’s mirror, and ETAI’s modified XOR, which means that the carry to the LSA (i.e., Cl pred) can also be differently speculated according to implementation of approximate logic. In other words, this is a generalized architecture in that the proposed design can replace the precise adder of any split architecture based approximate adder. The adder’s critical path delay tDSLLA can be given by

tDSLLA = max(tUSA + tupred, tLSA + tlpred + tERRL),

where tUSA, tLSA, and tERRL are the delays of the USA, LSA, and error recovery and reduction logic, respectively, and tu pred and tl pred are those of the carry predictions to the USA and LSA, respectively. Here, the critical path is dominated by each sub-adder’s delay because tERRL, tu pred, and tl pred are a few logic gate delay.

To reduce computation errors, we first leverage a carry prediction from the LSA to USA. Generally, approximate adders based on the split architecture employ an AND-based carry prediction, formulated by Cpred = Am−1Bm−1 ≜ Gm−1, for the accurate part where m – 1 is the inaccurate part’s MSB bit position. This reaches a prediction accuracy of approximately 75% under random inputs. In contrast, a lack of prediction leads to about 50%. An improved carry prediction scheme is used in our design instead of the AND-based one. Basically, we consider the LSA’s two MSB input pairs to predict a carry to the USA and this leads to a higher prediction accuracy of nearly 87.5%. Then, the carry Cu pred is given by

Cupred = Gn−u−1 + Pn−u−1Gn−u−2,

where Pn−u−1 = An−u−1 ⊕ Bn−u−1.Although the proposed adder includes a more accurate carry prediction, it still suffers from relatively worse error characteristics than the original precise adder due to approximately 12.5% of unsuccessful carry predictions. Therefore, we propose a novel error recovery and reduction logic that brings the overall approximation outputs closer to the accurate ones. First, the logic compares the actual carry output Cl out generated by the LSA and the carry Cu pred predicted from the LSA for the USA. If the two values matches, it does not require any recovery or reduction operation because the addition is always correct. Otherwise, the case is that of an unsuccessful carry prediction, the logic checks the USA’s LSB output S′n−u and then reduce the error distance by either adding the actual carry to the USA’s output or maximizing the LSA’s output.

# VERILOG

**Introduction**

# CHAPTER 4

**SOFTWARE DESCRIPTION**

In electronics, a hardware description language (HDL) is a specialized computer language used to program the structure, design and operation of electronic circuits and most commonly, digital logic circuits.

A hardware description language enables a precise, formal description of an electronic circuit that allows for the automated analysis, simulation, and simulated testing of an electronic circuit. It also allows for the compilation of an HDL program into a lower level specification of physical electronic components, such as the set of masks used to create an integrated circuit.

A hardware description language looks much like a programming language such as C, it is a textual description consisting of expressions, statements and control structures. One important difference between most programming languages and HDLs is that HDLs explicitly include the notion of time.

HDLs form an integral part of Electronic Design Automation (EDA) systems, especially for complex circuits, such as microprocessors.

## Motivation

Due to the exploding complexity of digital electronic circuits since the 1970s (see Moore's law), circuit designers needed digital logic descriptions to be performed at a high level without being tied to a specific electronic technology, such as CMOS or BJT. HDLs were created to implement register-transfer level abstraction, a model of the data flow and timing of a circuit.

There are two major hardware description languages: VHDL and Verilog. There are different types of description in them "dataflow, behavioral and structural".

## Structure of HDL

HDLs are standard text-based expressions of the structure of electronic systems and their behavior over time. Like concurrent programming languages,

HDL syntax and semantics include explicit notations for expressing concurrency. However, in contrast to most software programming languages, HDLs also include an explicit notion of time, which is a primary attribute of hardware. Languages whose only characteristic is to express circuit connectivity between a hierarchy of blocks are properly classified as netlist languages used in electric computer-aided design (CAD). HDL can be used to express designs in structural, behavioral or register-transfer-level architectures for the same circuit functionality in the latter two cases the synthesizer decides the architecture and logic gate layout.

HDLs are used to write executable specifications for hardware. A program designed to implement the underlying semantics of the language statements and simulate the progress of time provides the hardware designer with the ability to model a piece of hardware before it is created physically. Its executability that gives HDLs the illusion of being programming languages, when they are more precisely classified as specification languages or modeling languages. Simulators capable of supporting discrete-event (digital) and continuous-time (analog) modeling exist, and HDLs targeted for each are available.

## Comparison with Control-Flow Languages

It is certainly possible to represent hardware semantics using traditional programming languages such as C++, which operate on control flow semantics as opposed to data flow, although to function as such, programs must be augmented with extensive and unwieldy class libraries. Generally however, software programming languages do not include any capability for explicitly expressing time, and thus cannot function as hardware description languages. Before the introduction of System Verilog in 2002, C++ integration with a logic simulator was one of the few ways to use object-oriented programming in hardware verification. System Verilog is the first major HDL to offer object orientation and garbage collection.

Using the proper subset of hardware description language, a program called a synthesizer, or logic synthesis tool, can infer hardware logic operations from the language statements and produce an equivalent netlist of generic hardware

primitives [jargon] to implement the specified behavior. Synthesizers generally ignore the expression of any timing constructs in the text. Digital logic synthesizers, for example, generally use clock edges as the way to time the circuit, ignoring any timing constructs. The ability to have a synthesizable subset of the language does not itself make a hardware description language.

# DESIGN USING HDL

As a result of the efficiency gains realized using HDL, a majority of modern digital circuit design revolves around it. Most designs begin as a set of requirements or a high-level architectural diagram. Control and decision structures are often prototyped in flowchart applications, or entered in a state diagram editor. The process of writing the HDL description is highly dependent on the nature of the circuit and the designer's preference for coding style. The HDL is merely the 'capture language', often beginning with a high-level algorithmic description such as a C++ mathematical model. Designers often use scripting languages such as Perl to automatically generate repetitive circuit structures in the HDL language. Special text editors offer features for automatic indentation, syntax-dependent coloration, and macro-based expansion of entity/architecture/signal declaration.

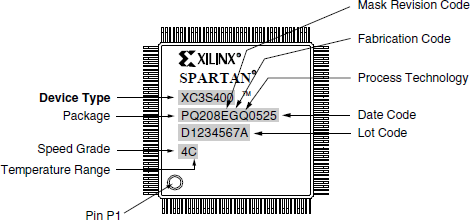
The HDL code then undergoes a code review or auditing. In preparation for synthesis, the HDL description is subject to an array of automated checkers. The checkers report deviations from standardized code guidelines, identify potential ambiguous code constructs before they can cause misinterpretation, and check for common logical coding errors, such as dangling[jargon] ports or shorted outputs. This process aids in resolving errors before the code is synthesized.

In industry parlance, HDL design generally ends at the synthesis stage. Once the synthesis tool has mapped the HDL description into a gate netlist, the netlist is passed off to the back-end stage. Depending on the physical technology (FPGA, ASIC gate array, ASIC standard cell), HDLs may or may not play a significant role in the back-end flow. In general, as the design flow progresses toward a physically realizable form, the design database becomes progressively more laden with technology-specific information, which cannot be stored in a

generic HDL description. Finally, an integrated circuit is manufactured or programmed for use.

# SPARTAN 3

The Spartan-3 family of Field-Programmable Gate Arrays is specifically designed to meet the needs of high volume, cost-sensitive consumer electronic applications. The eight-member family offers densities ranging from 60,000 to five million system gates. The Spartan-3 family builds on the success of the earlier Spartan-IIE family by increasing the amount of logic resources, the capacity of internal RAM, the total number of I/Os, and the overall level of performance as well as by improving clock management functions.

These Spartan-3 FPGA enhancements, combined with advanced process technology, deliver more functionality and bandwidth per dollar than was previously possible, setting new standards in the programmable logic industry. Because of their exceptionally low cost, Spartan-3 FPGAs are ideally suited to a wide range of consumer electronics applications including broadband access, home networking, display/ projection and digital television equipment. The Spartan-3 family is a superior alternative to mask programmed ASICs. FPGAs avoid the high initial cost, the lengthy development cycles, and the inherent inflexibility of conventional ASICs. Also, FPGA programmability permits design upgrades in the field with no hardware replacement necessary, an impossibility with ASICs.

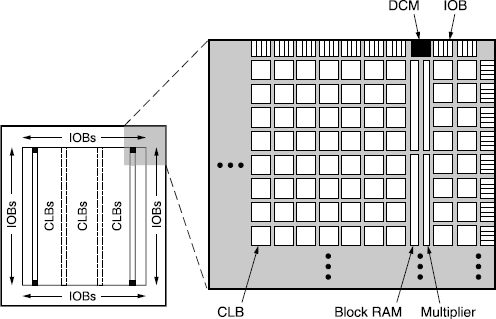
**Figure 4.1 Spartan-3 Package XC3S400-4PQ208C**

# ARCHITECTURALOVERVIEW

The Spartan-3 family architecture consists of five fundamental programmable functional elements:

1. Configurable Logic Blocks (CLBs) contain RAM-based Look-Up Tables (LUTs) to implement logic and storage elements that can be used as flip-flops or latches. CLBs can be programmed to perform a wide variety of logical functions as well as to store data.
2. Input/ Output Blocks (IOBs) control the flow of data between the I/O pins and the internal logic of the device. Each IOB supports bidirectional data flow plus 3- state operation. Twenty-six different signal standards, including eight high- performance differential standards are available. Double Data-Rate (DDR) registers are included. The Digitally Controlled Impedance (DCI) feature provides automatic on-chip terminations, simplifying board designs.
3. Block RAM provides data storage in the form of 18-K bit dual-port blocks.
4. Multiplier blocks accept two 18-bit binary numbers as inputs and calculate the product.
5. Digital Clock Manager (DCM) blocks provide self-calibrating, fully digital solutions for distributing, delaying, multiplying, dividing and phase shifting clock signals.

A ring of IOBs surrounds a regular array of CLBs. The XC3S60 has a single column of block RAM embedded in the array. Those devices ranging from the XC3S200 to the XC3S2000have two columns of block RAM. The XC3S4000 and XC3S6000 devices have four RAM columns. Each column is made up of several 18-Kbit RAM blocks; each block is associated with a dedicated multiplier.

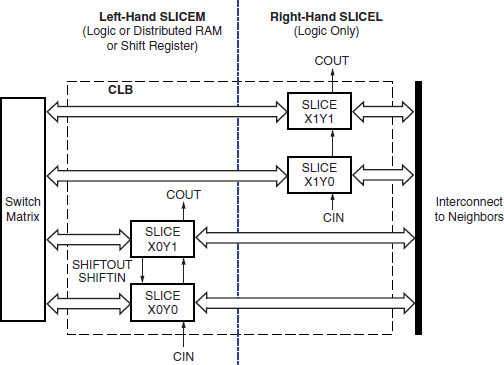


### Figure 4.2 Spartan-3 Architecture

The DCMs are positioned at the ends of the outer block RAM columns. The Spartan-3 family features a rich network of traces and switches that interconnect all five functional elements, transmitting signals among them. Each functional element has an associated switch matrix that permits multiple connections to the routing.

# CONFIGURABLE LOGIC BLOCK

The Configurable Logic Blocks (CLBs) constitute the main logic resource for implementing synchronous as well as combinatorial circuits. Each CLB comprises four interconnected slices as shown in Fig.6.3. These slices are grouped in pairs. Each pair is organized as a column with an independent carry chain.



### Figure 4.3 Arrangement of Slices within the CLB

The nomenclature that the FPGA Editor part of the Xilinx development software uses to designate slices is as follows: The letter ‘X’ followed by a number identifies columns of slices. The ‘X’ number counts up in sequence from the left side of the die to the right. The letter ‘Y’ followed by a number identifies the position of each slice in a pair as well as indicating the CLB row. The ‘Y’ number counts slices starting from the bottom of the die according to the sequence: 0, 1, 0, 1 (the first CLB row); 2, 3, 2, 3 (the second CLB row); etc. Fig. 6.3 shows the CLB located in the lower left-hand corner of the die. Slices X0Y0 and X0Y1 make up the column-pair on the left where as slices X1Y0 and X1Y1 make up the column- pair on the right. For each CLB, the term “left-hand” (or SLICEM) indicates the pair of slices labeled with an even ‘X’ number, such as X0, and the term “right- hand” (or SLICEL) designates the pair of slices with an odd ‘X’ number, e.g., X1.

## Download VirtualBox

* Download VirtualBox From The Site
* Link: https://[www.virtualbox.org/wiki/](http://www.virtualbox.org/wiki/) Downloads
* Launch the VirtualBox installer from Windows Explorer by double- clicking the self-extracting executable. Allow the installer to make changes to your computer, if so prompted.

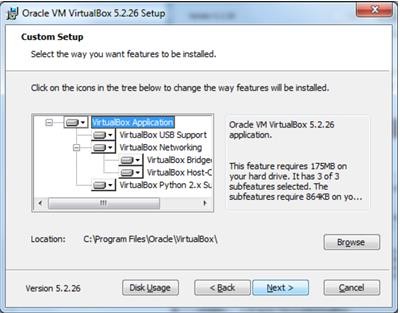


* Once the VirtualBox installation wizard appears, click the Next button.



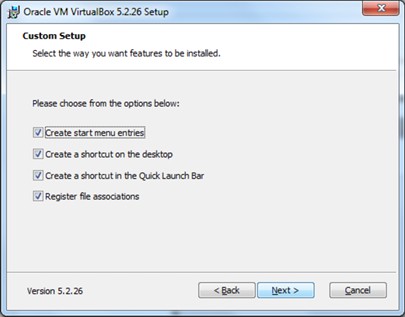
### Fig. 4.4 Setup of Virtual Box

* You may accept all the installation defaults, although you may wish to change the installation location on your development platform using the Browse button. If the options are acceptable, click the Next button.



### Fig. 4.5 Setup of Virtual Box Applications

* You may again accept the default options and click the Next button.



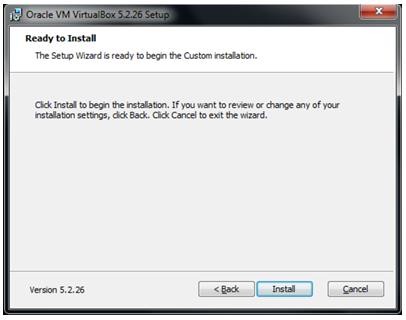
### Fig. 4.6 Custom Setup of Virtual Box

* Click the Yes button to continue with the installation wizard.



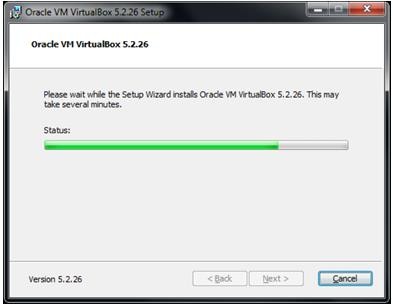
### . 4.7 Setup of Virtual Box Network Interfaces

* Click the Install button to load VirtualBox to your development system.



### Installing The Virtual Box

* During the installation you may receive prompts to authorize installation of various components. If prompted, allow the installer to make changes to your system, including installation of the USB interface and Network adapters.



### Installing In Processes

* If you are asked to install the Oracle Corporation Universal Serial Bus device driver, or Oracle Corporation Network Adapters/Network Service, choose to install them





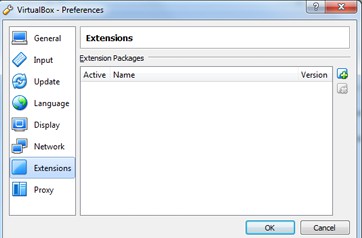
* 1. **Installing The Adapter And Serial Bus**
* Click the Finish button to complete the installation. Leave the checkbox enabled so VirtualBox will start after the installer finishes.



* 1. **Finishing Up The Installation**
* Once VirtualBox starts (you can also start it from the Desktop shortcut, or the Windows Start button), the Extension Pack must be added. From the main menu, select File > Preferences.



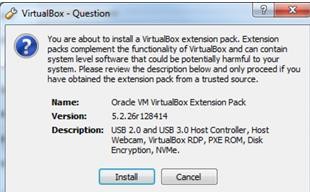
* 1. **Setting Up Virtual Box Preferences**
* Select Extensions. Right-click in the Extension Packages whitespace box and select Add Package



* 1. **Setting Up Extensions Of Virtual Box**
* Browse to the location where you downloaded the VirtualBox Extension Pack compatible with your VirtualBox version. Select the Extension Pack and click the Open button.

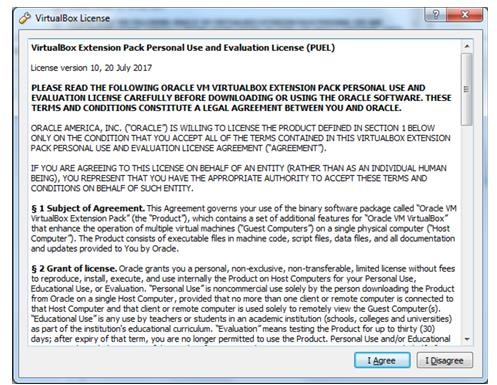


* 1. **Selecting The Extension Pack**
* Click the Install button to add the VirtualBox Extension Pack.



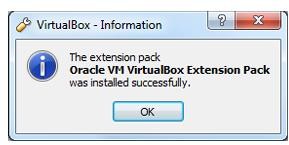
### Installing The Extension Pack

* Read the VirtualBox Extension Pack PUEL License to ensure you will not be in violation of the Oracle definition of Personal Use. See the VirtualBox Licensing Frequently Asked Questions for additional details. If you can accept the license conditions, scroll to the bottom of the agreement text box and click the I Agree button1. If prompted, allow the installer to make changes to your development system.



### Accepting The License Of Virtual Box

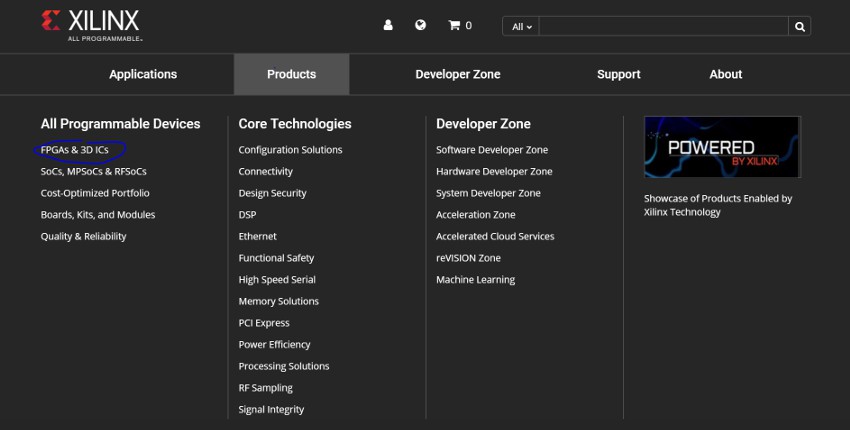
* Click the OK button to complete the installation.



* 1. **Completing The Installation**

## Download Xilinx ISE

* Download Xilinx ISE From The Site
* https:/[/www.xilinx.com/support/download/index.html/content/xilinx/en/download](http://www.xilinx.com/support/download/index.html/content/xilinx/en/download) Nav/design-tools/14\_7-windows.html



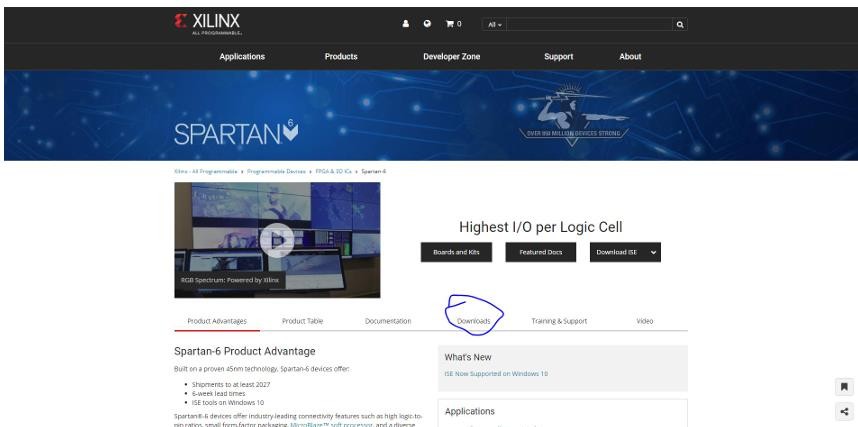
### Installing The Xilinx Software

* Whether you are a Mac or Windows user, download the compressed file. (Despite, if you on Mac, the fact that the download says “windows”) You may need to create an account with Xilinx and provide a company, I just used the campus address and contact info. If that works, skip the next slides and proceed to either “Windows Install” or “Mac Install”
* Navigate to Xilinx’s Products listing, and look for the product page for the “Spartan 6”



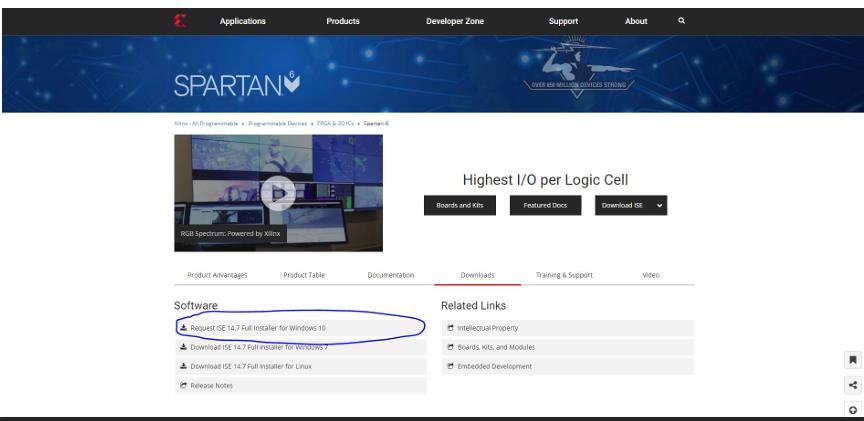
### Selecting The Specific Xilinx Software

* Look for the associated downloads tab



### Move To The Download Section

* Look for the Windows 10 Installer compressed download (tar or zip)
* Note: even if you are installing for Mac, we will use the Windows 10 install files

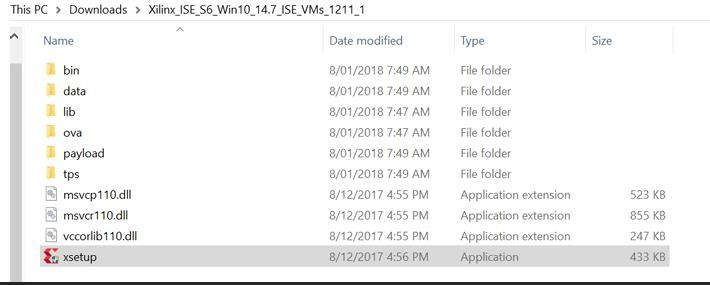


* 1. **Downloading Latest Version Xilinx**

**Software**

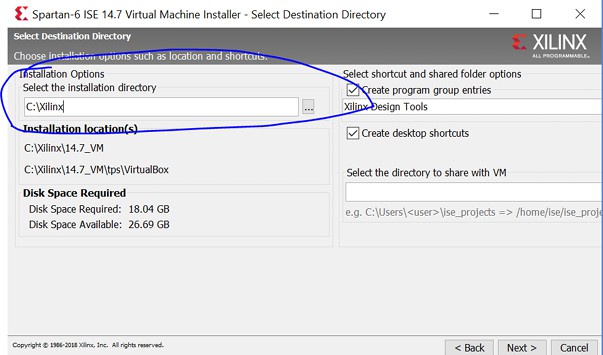
## Installing Xilinx In Virtual Box

* Just unzip the installer and click “xsetup”
* It’s pretty straightforward, but there’s a couple options that you might want to pay attention to (see next two slides before install)



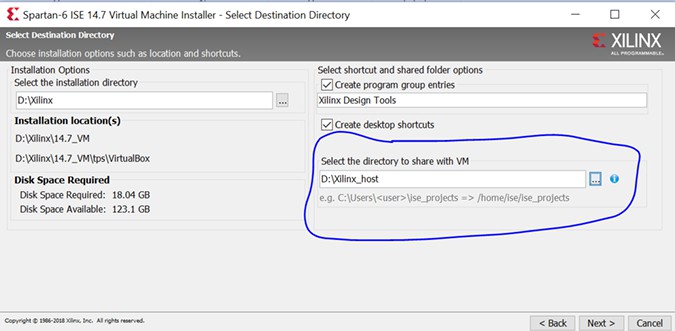
### Setting Up Xilinx Software In Virtual Vox

* At this stage, you can select the install directory. Xilinx takes up a large swath of memory so it’s a good idea to set this to a bigger hard drive



### Allocating The Location For Installing

* You can set up a folder to be visible from both your (host) operating system and the virtual machine’s (guest) operating system. This is good for quickly moving files around. (See Usage Notes)



### Giving Access To The Windows Folders

* You may need to enable hardware virtualization on your computer
* For that, you will need to restart and enter the BIOS and find the setting “virtualization” and enable it



* 1. **Enabling The Virtualization Option**
  2. **Advantages**

# CHAPTER 5

**ADVANTAGES AND APPLICATIONS**

* + - **Low-Power IoT Devices –** Enables efficient data processing in battery-powered IoT sensors and edge computing applications.
    - **Autonomous Systems & Robotics –** Helps in efficient real-time processing for object detection, motion planning, and AI-driven decision-making.
    - **Neural Networks & Deep Learning –** Optimizes multiplication operations in neural network training and inference for faster, energy-efficient computations.
    - **Wireless Communication Systems –** Improves efficiency in error-tolerant computations like modulation, demodulation, and encoding in 5G and other communication technologies.
    - **Image & Video Processing –** Allows configurable accuracy for tasks like image filtering, compression, and enhancement, saving energy in multimedia applications.

## Applications

* + - **Low-Power IoT Devices** – Enables efficient data processing in battery-powered IoT sensors and edge computing applications.
    - **Autonomous Systems & Robotics** – Helps in efficient real-time processing for object detection, motion planning, and AI-driven decision-making.
    - **Neural Networks & Deep Learning** – Optimizes multiplication operations in neural network training and inference for faster, energy-efficient computations.
    - **Wireless Communication Systems** – Improves efficiency in error-tolerant computations like modulation, demodulation, and encoding in 5G and other communication technologies.

# CHAPTER 6

**RESULTS AND DISCUSSION**

The proposed accuracy-configurable approximate multiplier was coded in Verilog and simulated using Xilinx ISE 12.1 with the Spartan-3E FPGA platform for evaluation. This evaluation focused on key performance metrics, including slice count, lookup tables (LUT) utilization, and delay to assess the hardware efficiency and speed of the design. The two multiplier variants—one optimized for speed and power efficiency and the other for balanced accuracy and performance—were tested separately to compare their behavior under different configurations.

## Slice Count and Area Efficiency

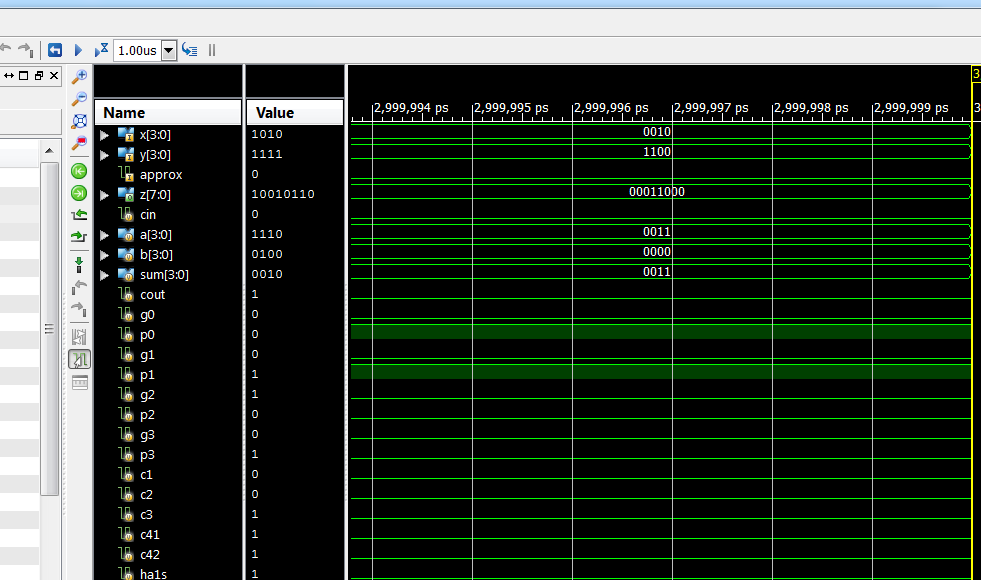
The design’s slice count was significantly lower than traditional multipliers due to the novel dual sub-adder architecture, which effectively reduces the hardware complexity by splitting the precise adder into two approximate sub- adders. This reduction in slice count results in a more area-efficient multiplier, making it suitable for resource-constrained environments like embedded systems. The first variant, which maximizes approximation, demonstrated a further decrease in slice usage compared to the more accurate variant.

## LUT Utilization

LUT usage was measured to analyze how efficiently the multiplier maps onto the FPGA’s configurable logic. The approximate adder structure led to a reduction in LUT count, as it avoids complex addition stages typically required in precise designs. The accuracy configuration module also played a role in maintaining optimal LUT usage by dynamically managing approximation based on the application's requirements. The less accurate variant showed a more significant reduction in LUT usage, benefiting applications prioritizing power and speed.

## Delay Analysis

The delay performance was also measured, showing a noticeable reduction compared to conventional multipliers. The dual sub-adder approach helps to minimize critical path delays, allowing faster propagation of partial sums. The high-speed variant achieved the lowest delay, making it highly suitable for applications where rapid computation is critical. However, the balanced variant showed slightly higher delay due to additional error recovery steps but still

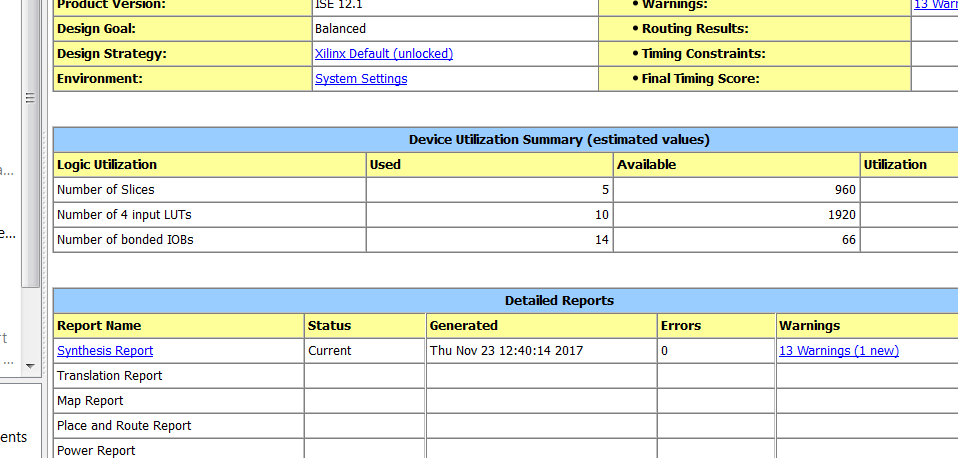
performed efficiently within acceptable limits.The Spartan-3E FPGA results thus demonstrate that the proposed approximate multiplier offers substantial improvements in area, power efficiency, and speed while retaining configurable accuracy, making it an ideal choice for error-resilient applications requiring high performance with minimal resource utilization. The findings indicate that this design could be a practical solution for real-world scenarios where computational efficiency outweighs the need for exact precision.

### Fig6.1. simulation result

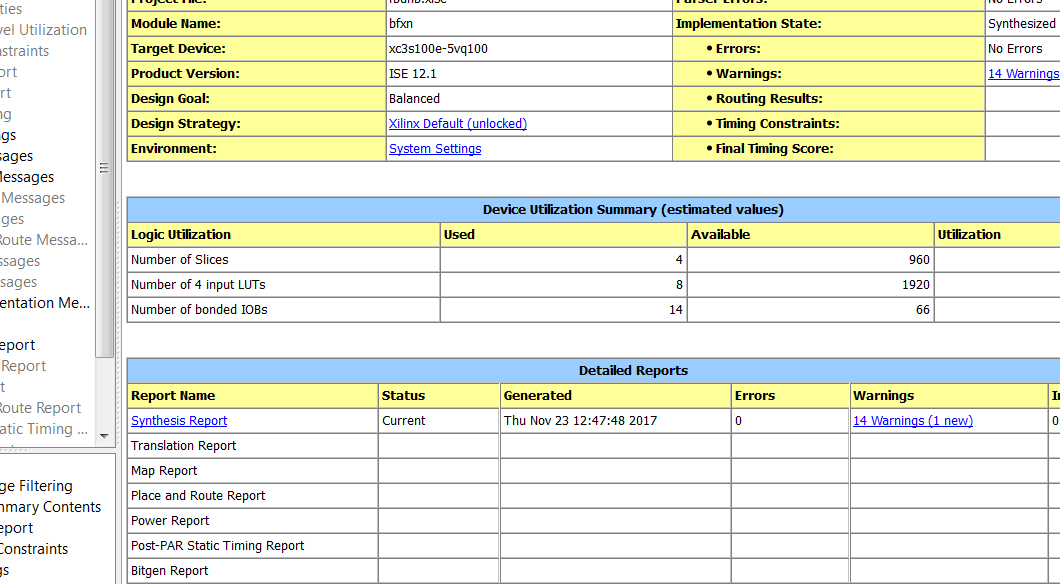
****

**Fig.6.2 simulation result**

The existing and proposed has been simulated and the synthesis report can be obtained by using Xilinx ISE 12.1i. The various parameters used for computing existing and proposed systems with Spartan-3 processor are given in the table 7.1.

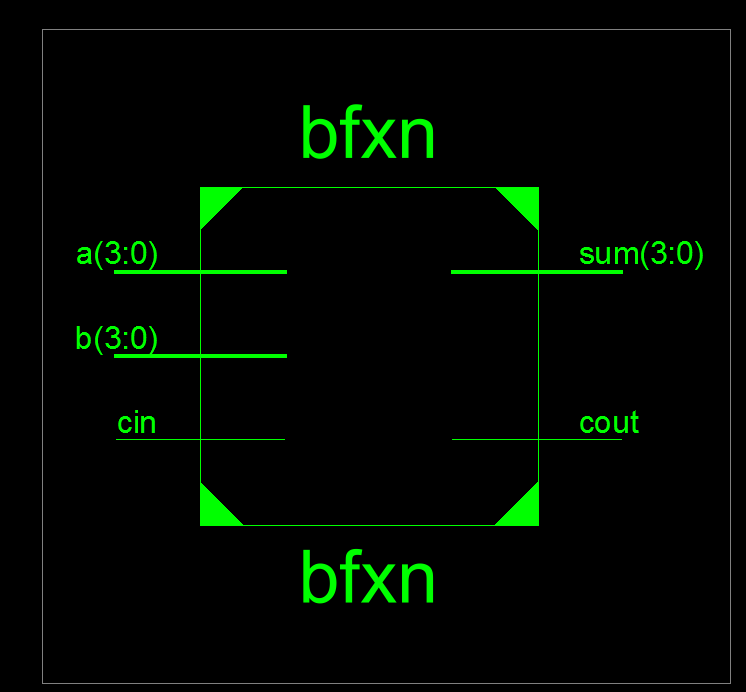


### Fig.6.3 Design Summary of Existing

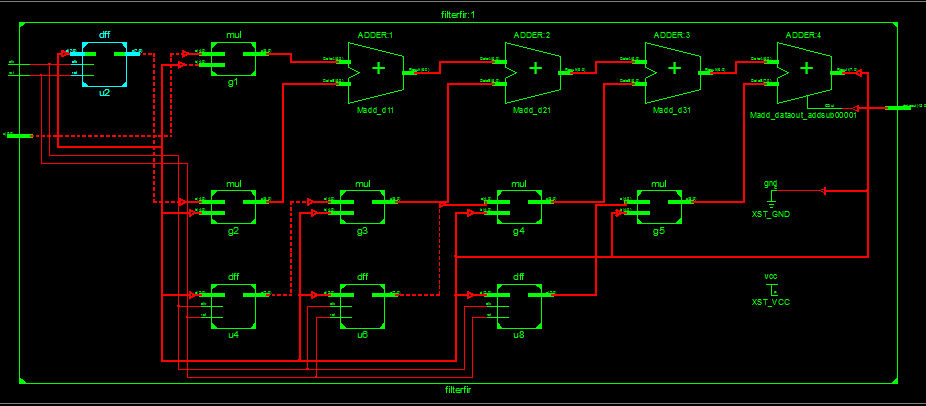
****

**Fig.6.4 Design Summary of Proposed**

After performing the synthesize process, the RTL schematic has been created automatically based on the functionality. The routing between the different cells can be viewed clearly by this schematic.

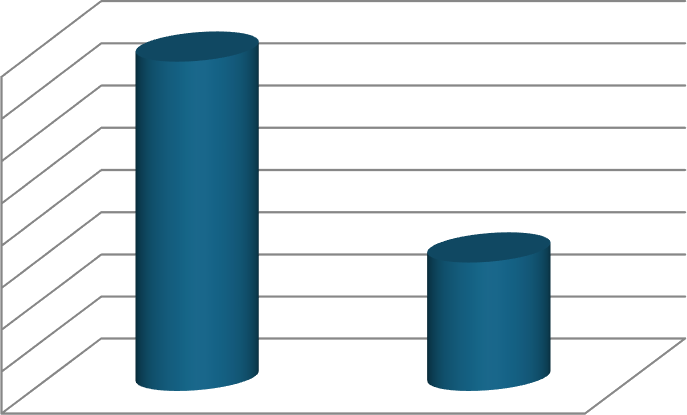


### Fig.6.5 RTL Schematic

****

**Fig.6.6 Gate level Netlist**

The Figure given below is shown that there is a considerable reduction in time and area based on the implementation results which have been done by using Spartan-3 processor. The proposed algorithm significantly reduces area consumption when compared to the existing system.



**Delay**

9

8.8

8.6

8.4

8.2

8

7.8

7.6

7.4

Delay

Existing Proposed

**Fig.6.7 performance results**

# CHAPTER 7 CONCLUSION AND FUTURE SCOPE

## Conclusion

In conclusion, this work presents an innovative design for an accuracy- configurable approximate multiplier aimed at enhancing power efficiency and performance in error-resilient applications. By introducing a novel dual sub-adder based approximate adder, the system effectively reduces latency and hardware complexity, making it well-suited for real-time, low-power applications such as image processing, signal processing, and machine learning. The proposed error recovery and reduction technique successfully mitigates the accuracy loss introduced by the approximation, enabling the multiplier to deliver reliable results even in scenarios where a high degree of precision is not critical.The system's adaptability is further enhanced by the inclusion of two multiplier variants, each optimized to balance accuracy and performance for different application requirements. Extensive evaluation using Xilinx 12.1

## Future Scope

* + 1. Optimization for Emerging Technologies
* Implementing the design on advanced process nodes (e.g., 5nm, 3nm) to further improve power efficiency.
  + 1. Integration with AI and Machine Learning
* Enhancing the multiplier for deep learning accelerators where configurable accuracy can improve efficiency.
  + 1. Application in High-Performance Computing (HPC)
* Adapting the design for parallel computing architectures to boost efficiency in scientific and cryptographic computations.
  + 1. Extension to Approximate Computing
* Enhancing approximate computing techniques by introducing adaptive error recovery mechanisms.

## References

1. W. Liu et al., "Design and Analysis of Approximate Redundant Binary Multipliers," in IEEE Transactions on Computers, vol. 68, no. 6, pp. 804-819, 1 June 2019, doi: 10.1109/TC.2018.2890222.
2. Y. Zhang et al., "An Energy-Efficient Binary-Interfaced Stochastic Multiplier Using Parallel Datapaths," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 31, no. 9, pp. 1439-1443, Sept. 2023, doi: 10.1109/TVLSI.2023.3247739.
3. N. Amirafshar, A. S. Baroughi, H. S. Shahhoseini and N. TaheriNejad, "Carry Disregard Approximate Multipliers," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 70, no. 12, pp. 4840-4853, Dec. 2023, doi: 10.1109/TCSI.2023.3306071.
4. S. Venkatachalam and S. -B. Ko, "Design of Power and Area Efficient Approximate Multipliers," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 25, no. 5, pp. 1782-1786, May 2017, doi: 10.1109/TVLSI.2016.2643639.
5. D. Esposito, A. G. M. Strollo, E. Napoli, D. De Caro and N. Petra, "Approximate Multipliers Based on New Approximate Compressors," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 65, no. 12, pp. 4169-4182, Dec. 2018, doi: 10.1109/TCSI.2018.2839266.
6. F. Sabetzadeh, M. H. Moaiyeri and M. Ahmadinejad, "An Ultra-Efficient Approximate Multiplier With Error Compensation for Error-Resilient Applications," in IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 70, no. 2, pp. 776-780, Feb. 2023, doi: 10.1109/TCSII.2022.3215065.
7. M. Zhang, S. Nishizawa and S. Kimura, "Area Efficient Approximate 4–2 Compressor and Probability-Based Error Adjustment for Approximate Multiplier," in IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 70, no. 5, pp. 1714-1718, May 2023, doi: 10.1109/TCSII.2023.3257852.
8. Z. Aizaz and K. Khare, "Area and Power Efficient Truncated Booth Multipliers Using Approximate Carry-Based Error Compensation," in IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 69, no. 2, pp. 579-583, Feb. 2022, doi: 10.1109/TCSII.2021.3094910.
9. G. Park, J. Kung and Y. Lee, "Simplified Compressor and Encoder Designs for Low-Cost Approximate Radix-4 Booth Multiplier," in IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 70, no. 3, pp. 1154-1158, March 2023, doi: 10.1109/TCSII.2022.3217696.
10. H. Waris, C. Wang and W. Liu, "Hybrid Low Radix Encoding-Based Approximate Booth Multipliers," in IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 67, no. 12, pp. 3367-3371, Dec. 2020, doi: 10.1109/TCSII.2020.2975094.
11. W. Liu, T. Zhang, E. McLarnon, M. O’Neill, P. Montuschi and F. Lombardi, "Design and Analysis of Majority Logic-Based Approximate Adders and Multipliers," in IEEE Transactions on Emerging Topics in Computing, vol. 9, no. 3, pp. 1609-1624, 1 July-Sept. 2021, doi: 10.1109/TETC.2019.2929100.
12. W. Liu, L. Qian, C. Wang, H. Jiang, J. Han and F. Lombardi, "Design of Approximate Radix-4 Booth Multipliers for Error-Tolerant Computing," in IEEE Transactions on Computers, vol. 66, no. 8, pp. 1435-1441, 1 Aug. 2017, doi: 10.1109/TC.2017.2672976.
13. V. Leon, G. Zervakis, D. Soudris and K. Pekmestzi, "Approximate Hybrid High Radix Encoding for Energy-Efficient Inexact Multipliers," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 26, no. 3, pp. 421-430, March 2018, doi: 10.1109/TVLSI.2017.2767858.
14. M. Masadeh, O. Hasan and S. Tahar, "Machine-Learning-Based Self-Tunable Design of Approximate Computing," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 29, no. 4, pp. 800-813, April 2021, doi: 10.1109/TVLSI.2021.3056243.
15. W. Pan, Y. Yu, C. Tang, N. Yin and Z. Yu, "Low-Power, High-Speed, and Area- Efficient Multiplier Based on the PTL Logic Style," in IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 71, no. 7, pp. 3538-3542, July 2024, doi: 10.1109/TCSII.2024.3366169.
16. M. H. Haider, H. Zhang and S. -B. Ko, "Decoder Reduction Approximation Scheme for Booth Multipliers," in IEEE Transactions on Computers, vol. 73, no. 3, pp. 735-746, March 2024, doi: 10.1109/TC.2023.3343093.
17. J. Vafaei and O. Akbari, "HPR-Mul: An Area and Energy-Efficient High-Precision Redundancy Multiplier by Approximate Computing," in IEEE Transactions on

Very Large Scale Integration (VLSI) Systems, vol. 32, no. 11, pp. 2012-2022, Nov. 2024, doi: 10.1109/TVLSI.2024.3445108.

**ANNEXURE-A**

### SOURCE CODE

module multiplier\_16bit1602(product, a, b); input [15:0] a, b;

output [31:0] product;

wire [7:0] partial\_sum; wire carry;

reg [31:0] temp\_product; integer i;

correcexst16 adder(.sum(partial\_sum),.car(carry),.a(a),

.b(temp\_product[15:0]), .cin(1'b0)); always @(\*) begin

temp\_product = 32'b0;

for (i = 0; i < 16; i = i + 1) begin if (b[i] == 1'b1) begin

temp\_product = temp\_product + (a << i); end

end end

assign product = temp\_product; endmodule

module correcexst16(sum,car,a,b,cin); input [15:0]a,b;

input cin;

output [15:0]sum; reg [15:0]sum; output car;

reg car;

reg car1a,car1b, car2a,car2a1, car3a,car3a1,car3a2,car3b, car4a,car4a1,car4a2,car4a3,

car5a,car5a1,car5a2,car5a3,car5a4;

reg sum1a,sum1b, sum2a,sum2a1, sum3a,sum3a1,sum3a2, sum4a,sum4a1,sum4a2,sum4a3,

sum5a,sum5a1,sum5a2,sum5a3,sum5a4, sum6a,sum6a1,sum6a2,sum6a3,sum6a4,sum6a5, sum7a,sum7a1,sum7a2,sum7a3,sum7a4,sum7a5,sum7a6, sum8a,sum8a1,sum8a2;

always @(\*) begin

sum1a = a[0]^b[0];

car1a = (a[0]&b[0])|(b[0]&cin); sum1b = a[1]^b[1];

car1b = (a[1]&b[1])|(b[1]&car1a); sum2a = a[2]^b[2];

car2a = (a[2]&b[2])|(b[2]&car1b); sum2a1 = a[3]^b[3];

car2a1 = (a[3]&b[3])|(b[3]&car2a); sum3a = a[4]^b[4];

car3a = (a[4]&b[4])|(b[4]&car2a1); sum3a1 = a[5]^b[5];

car3a1 = (a[5]&b[5])|(b[5]&car3a); sum3a2 = a[6]^b[6];

car3a2 = (a[6]&b[6])|(b[6]&car3a1); sum4a = a[7]^b[7];

car4a = (a[7]&b[7])|(b[7]&car3a2); if( a[7]==b[7]==1)

begin

sum[0]= sum1a; sum[1]= sum1b; sum[2] = sum2a; sum[3] = sum2a1;

sum[4] = sum3a; sum[5] = sum3a1; sum[6] = 0;

sum[7] = 1; end

else

begin sum[0]= 1;

sum[1]= 1;

sum[2] = 1;

sum[3] = 1;

sum[4] = 1; sum[5] = sum3a1; sum[6] = sum3a2; sum[7] = sum4a; end

sum[8] = a[8]^b[8]^car4a;

car4a1 = (a[8]&b[8])|(b[8]&car4a)|(car4a&a[8]); sum[9] = a[9]^b[9]^car4a1;

car4a2 = (a[9]&b[9])|(b[9]&car4a1)|(car4a1&a[9]); sum[10] = a[10]^b[10]^car4a2;

car4a3 = (a[10]&b[10])|(b[10]&car4a2)|(car4a2&a[10]); sum[11] = a[11]^b[11]^car4a3;

car5a = (a[11]&b[11])|(b[11]&car4a3)|(car4a3&a[11]); sum[12] = a[12]^b[12]^car5a;

car5a1 = (a[12]&b[12])|(b[12]&car5a)|(car5a&a[12]); sum[13] = a[13]^b[13]^car5a1;

car5a2 = (a[13]&b[13])|(b[13]&car5a1)|(car5a1&a[13]); sum[14] = a[14]^b[14]^car5a2;

car5a3 = (a[14]&b[14])|(b[14]&car5a2)|(car5a2&a[14]); sum[15] = a[15]^b[15]^car5a3;

car5a4 = (a[15]&b[15])|(b[15]&car5a3)|(car5a3&a[15]); sum[8] = sum4a1;

sum[9] = sum4a2; sum[10] = sum4a3; sum[11] = sum5a; sum[12] = sum5a1; sum[13] = sum5a2; sum[14] = sum5a3; sum[15] = sum5a4; car=car5a4;

end endmodule

**ANNEXURE-B**

**ANNEXURE-C**