Design and Implementation of a Novel Approximate Parallel Prefix Adders for Image Processing Applications

PROJECT REPORT

Submitted in the fulfilment of the requirements for

the award of the degree of

Bachelor of Technology
in
Electronics and Communication Engineering

DANABONIA HEMANTH [201FA05051]

DANTLA SUDHAKAR REDDY [201FA05093]

TUMMALA PRUDHVI [211LA05018]

GHANTASALA JASWANTHKUMAR [211LA05025]

Under the Esteemed Guidance of

Dr. M.Sarada

Professor

Department of ECE

(ACCREDITED BY **NAAC** WITH **"A^{+}"** GRADE)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
(ACCREDITED BY **NBA**)

VIGNAN'S FOUNDATION FOR SCIENCE, TECHNOLOGY AND RESEARCH

(Deemed to be University)

Vadlamudi, Guntur, Andhra Pradesh, India -522213

May 2024

CERTIFICATE

This is to certify that project report entitled **"Design and Implementation of a Novel Approximate Parallel Prefix Adders for Image Processing Applications"** that is being submitted by Danabonia Hemanth [20IFA05051], Dantla Sudhakar Reddy [20IFA05093], Tummala Prudhvi [21ILA05018] and Ghantasala Jaswanth Kumar [21ILA05025] in fulfilment for the award of B. Tech degree in Electronics and Communication Engineering, Vignan's Foundation for Science Technology and Research University, is a record of bonafide work carried out by them under the guidance of Dr.M. Sarada, Professor of ECE Department.

23/05/2024

Signature of the guide

Dr. M. Sarada

Professor

23/5/24.

Signature of Head of the Department

Dr. T. Pitchaiah, M.E, Ph.D, MIEEE, FIETE

Professor & HoD ECE

DECLARATION

We hereby declare that the project report entitled **"Design and Implementation of a Novel Approximate Parallel Prefix Adders for Image Processing Applications"** is being submitted to Vignan's Foundation for Science, Technology and Research (Deemed to be University) in fulfilment for the award of B. Tech degree in Electronics and Communication Engineering. The work was originally designed and executed by us under the guidance of **Dr. M. Sarada** at the Department of Electronics and Communication Engineering, Vignan's Foundation for Science Technology and Research (Deemed to be University) and was not a duplication of work done by someone else. We hold the responsibility of the originality of the work incorporated into this project report.

Signature of the candidates

DANABOINA HEMANTH	(201FA05051)	D. Hemanth
DANTLA SUDHAKAR REDDY	(201FA05093)	D. Sudhakar Reddy
TUMMALA PRUDHVI	(211LA05018)	T. Prudhvi
GHANTASALA JASWANTH KUMAR	(211LA05025)	G. Jaswanth

ACKNOWLEDGEMENT

The satisfaction that comes from successfully completing any task would be incomplete without acknowledging the people who made it possible, whose ongoing guidance and encouragement have been essential to the achievement.

We are greatly indebted to **Dr. M. Sarada**, my revered guide and Professor in the Department of Electronics and Communication Engineering, VFSTR (Deemed to be University), Vadlamudi, Guntur, for his valuable guidance in the preparation of this project report. He has been a source of great inspiration and encouragement to us. He has been kind enough to devote considerable amount of his valuable time in guiding us at every stage. This is our debut, but we are sure that we are able to do many more such studies, under the lasting inspiration and guidance given by respectable guide.

We would also like to thank to **Dr. T. Pitchaiah**, Head of the Department, ECE for his valuable suggestion.

We would like to specially thank, **Dr. N. Usha Rani**, Dean, School of Electrical, Electronics and Communication Engineering for her help and support during the project work.

We thank our project coordinators **Dr. Satyajeet Sahoo, Dr. Arka Bhattacharyya, Mr. Abhishek Kumar** and **Mr. M. Vamsi Krishna** for continuous support and suggestions in scheduling project reviews and verification of the report. Also, thank to supporting staff of ECE Department for their technical support for timely completion of project.

We would like to express our gratitude to **Dr. P. Nagabhusan**, Vice-Chancellor, VFSTR (Deemed to be University) for providing us the greatest opportunity to have a great exposure and to carry out the project.

Finally, we would like to thank our parents and friends for the moral support throughout the project work.

Name of the Student

DANABOINA HEMANTH (201FA05051)

DANTLA SUDHAKAR REDDY (201FA05093)

TUMMALA PRUDHVI (211LA05018)

GHANTASALA JASWANTH KUMAR (211LA05025)

Abstract

Approximate parallel prefix adders (AxPPA) are a type of circuit that can perform addition operations on binary numbers with high speed and low power consumption with less area. These circuits provide approximate results, which means that they sacrifice accuracy for efficiency. This trade off makes them ideal for use in applications where speed is more important than precision. In recent years, there has been a growing interest in the development of these circuits, as they have the potential to revolutionize the field of digital signal processing. In this project, we will provide an overview of approximate parallel prefix adders and methods for reducing the error rate of AxPPA by modification of the architectures for the following PPA adders: approximate Brent–Kung (AxBKPPA), approximate Kogge–Stone (AxKSPPA), Ladner-Fischer (AxLFPPA), Knowles (AxKWPPA), Han Carlson (AxHCPPA) and Sklansky (AxSKPPA). The proposing adders aims at reduce the area, power dissipation, delay and error rate. These will be designed, simulated and implemented with Verilog HDL (IEEE 1364-2005), Xilinx Vivado software tool (Xilinx Vivado2018.1) and Digilent Artix7 FPGA board (XC7A35TICSG324-1L) respectively. The adders are verified by implementing them in image processing applications.

Major Design (Final Year Project Work) Experience Information

Student Group	D.HEMANTH (201FA05051)	D.SUDHAKAR REDDY (201FA05093)	T. PRUDHVI (211LA05018)	G. JASWANTH KUMAR (211LA05025)
Project Title	Design and Implementation of Novel Approximate Parallel Prefix Adders for Image Processing Applications			
Program Concentration Area	Design and implementation of approximate parallel prefix adders using Verilog HDL and FPGA.			
Program Concentration Area	Implementation of image processing applications.			
Constraints - Examples				
Economic	Fixed budget			
Environmental	Friendly			
Sustainability	To save power, delay and area by using different approximate parallel prefix adder techniques.			
Manufacturability	No need, direct hardware kit available			
Ethical	Followed the standard professional ethics			
Health and Safety	Guidelines are followed			
Social	Applicable for computing architectures			
Politicalc0020	None			
Other	Nil			
Standards				
1.IEEE 1364-2005	Verilog Hardware Description Language (HDL) used in digital design.			
2.IEEE 1801-2009/2018	Unified Power Format (UPF).			
3.IEEE 1588	Precision Time Protocol (PTP), used for synchronization of clocks in a computer network.			
4.Xilinx Vivado 2018.1	FPGA design and implementation, developed by Xilinx			
5.Xilinx Artix7 FPGA board (XC7A35TICSG324-1L)	System-on-Chip device that integrates FPGA fabric with ARM Cortex-A9 processors, providing a versatile platform for embedded systems development.			
6. IEEE 1149.1-1990	Joint Test Action Group(JTAG)			
Previous Course Required for the Major Design Experience	1.Digital system design 2.FPGA Based Design 3.Digital design through Verilog			

Supervisor 23/05/2024 Project co-ordinator Head of the department ECE 23/5/24.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS AND ABBREVIATIONS

Ax	Approximate
AC	Approximate Computing
AxPPA	Approximate Parallel Prefix Adders
AxBK	Approximate Brent Kung
AxKS	Approximate Kogge Stone
AxSK	Approximate Sklansky
AxLF	Approximate Ladner Fischer
AxHC	Approximate Han Carlson
BK	Brent Kung
FIL	FPGA IN LOOP
FPGA	Field Programmable Gate Array
HC	Han Carlson
KS	Kogge Stone
KW	Knowles
LF	Ladner Fishner
PPA	Parallel Prefix Adder

CHAPTER-1

INTRODUCTION

1.1 INTRODUCTION

Parallel Prefix Adders in VLSI are a class of digital circuits and systems, adders play a vital role in performing fast arithmetic operations is increasing day by day. They are highly efficient provide performance in terms of speed, area and power consumption. they are widely used in VLSI circuits such as microprocessors, digital signal processors etc and some of the design methodologies used for building these adders are the brent kung, kogge stone, sklansky, Ladner Fisher, Knowles, and Han Carlson adder. Approximate parallel prefix adders instead of exact parallel prefix adders for certain applications where exactness is not critical and some amount of error can be tolerated. These approximate adders are designed to provide approximate output with lower area and power consumption and higher speed compared to exact parallel prefix adders.

Approximate computing is a computing paradigm that trades off accuracy for efficiency. Instead of always providing precise results, approximate computing techniques aim to deliver results that are "approximately correct" within certain acceptable bounds. This approach can be particularly useful in scenarios where exact accuracy is not critical or where there are resource constraints such as time, energy, or hardware limitations. Some common techniques used in approximate computing include Reduced Precision Arithmetic, Algorithmic Approximations, Selective Precision, Early Termination, Speculative Execution, Error-Tolerant Data Structures. It is particularly relevant in domains such as signal processing, machine learning, multimedia applications, and scientific computing, where small errors in computation can often be tolerated without significantly impacting the overall result or user experience. However, it's essential to carefully analyze the impact of approximation on the specific application to ensure that the introduced errors are within acceptable bounds.

1.2 MOTIVATION

Approximate computing (AC) is a method that trades off accuracy for better performance and energy efficiency. It takes advantage of the fact that many applications are resilient to imprecision and soft errors. Some advantages of approximate computing include, Improved network performance, Energy efficiency, Image processing etc. AC can be an innovative method for image processing, with the advantages of lower power and high accuracy. Addition is one of the important and basic requirements in AC with more speed. To get more speed Parallel prefix adders are used. So, this project aims to design and implement Approximate Parallel prefix adders (AxPPA).

1.3 OBJECTIVES

To design and implement:

- AxPPA with less Area
- AxPPA with more speed
- AxPPA with less error rate
- Implementing Image processing application with more PSNR

1.4 TOOLS AND STANDARDS

Tools and standards are:

- IEEE 1364-2005 is the standard for Verilog Hardware Description Language (HDL) used in digital design.
- Xilinx Vivado 2018.1 is a software tool suite for FPGA design and implementation, developed by Xilinx.
- IEEE 1149.1-1990 Joint Test Action Group(JTAG)
- IEEE 1801-2009/2018, also known as the Unified Power Format (UPF), is a standard for specifying power intent in electronic design.
- IEEE 1588 is the standard for Precision Time Protocol (PTP), used for synchronization of clocks in a computer network.
- Xilinx Artix7 FPGA board (XC7A35TICSG324-1L) is a System-on-Chip device that integrates FPGA fabric with ARM Cortex-A9 processors, providing a versatile platform for embedded systems development.

CHAPTER-2

LITERATURE REVIEW

2.1 INTRODUTION

In the realm of Approximate VLSI, parallel prefix adders play a significant role in processing chips as they are utilized to add two large binary values and perform the addition operation. The primary objectives for this type of adder are to minimize the area, improve speed, and reduce the error rate. In order to tackle the challenges faced in the Exact PPA domain, enhancements have been made to enhance efficiency and decrease delay. The introduction of the current PPA adder aimed to achieve these goals by minimizing the area, reducing delay, and error rates. However, to further optimize the delay and error rate, the development of the Approximate PPA took place. This innovative solution aims to deliver superior results at the output by further reducing delay while maintaining a minimal error rate. The computation results, which are nearly accurate, are evaluated as approximate, thus confirming their suitability for practical use in Image Processing Applications. Approximate computing, also known as AxC, is a novel design approach that aims to enhance efficiency throughout the computing stack by leveraging the natural fault tolerance of various applications. AxC introduces accuracy, which refers to the quality of outcomes, as a fresh explicit aspect for balancing design enhancements. This strategy can significantly reduce the VLSI circuit area and energy usage for computations.

2.2 EXISTING APPROXIMATE PPA

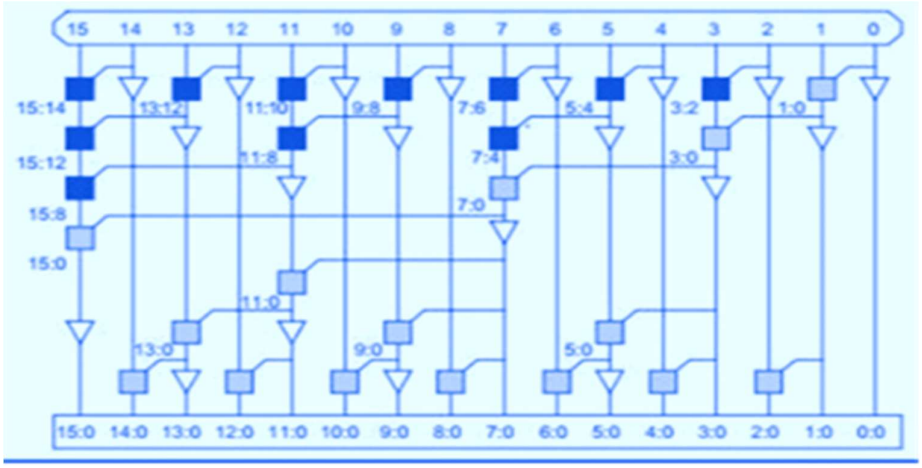

Fig. 2.2.1 Exact 16-Bit Brent Kung Adder

The parallel arithmetic operations performed by the Brent-Kung adder are implemented using a tree structure. Richard Peirce Brent and Hsiang Te Kung first proposed it in 1982. This is a great choice for low-power designs as it is designed to reduce chip space and facilitate manufacturing. The gray and black cells make up the Brent-Kung adder, with each black cell having two AND gates and one OR gate. Compared to other adder topologies, it has greater regularity, which reduces cable

clutter and improves performance. Figure 2.2.1 shows the structure of the 16-bit Brent Kung adder. Calculating the carry from the least significant bit adder (LSB) to the most significant bit adder (MSB) determines the critical route in the BK adder, which is why one tries to shorten the critical path so that the storage unit can reach the MSB.

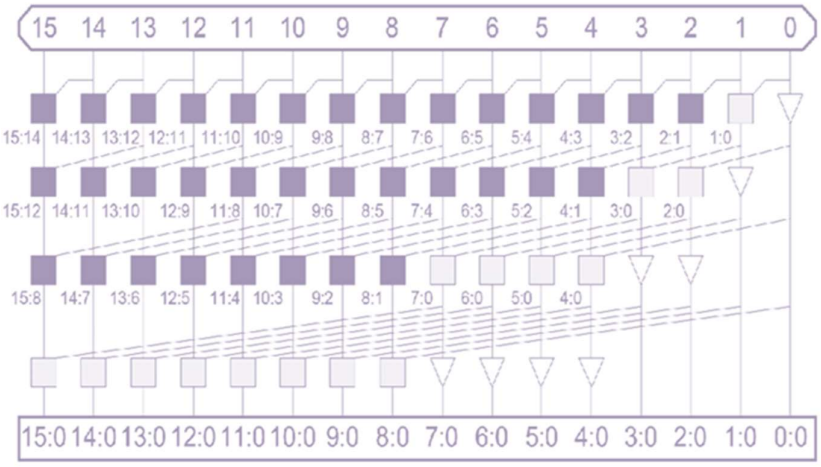

Fig. 2.2.2 Exact 16-Bit Kogge Stone Adder

Kogge Stone Adder is a parallel-prefix shape carry look-ahead adder. It was once developed via Peter M. Kogge and Harold S. Stone which they posted in 1973. KS adder is a fast adder layout as it generates carry signal in low delay and has the satisfactory overall performance in VLSI implementations. Kogge-Stone adder is broadly used in excessive overall performance 32-bit, 64-bit, as it reduces the crucial direction to great extent.Figure 2.2.2 represents the 16-bit Kogge Stone adder structure, which has fanout of 2 at each stage. There are a less number of stages with high complexity logical operational architecture. KS Adder is broadly used in excessive overall performance addition devices with area efficiency and less time delay.

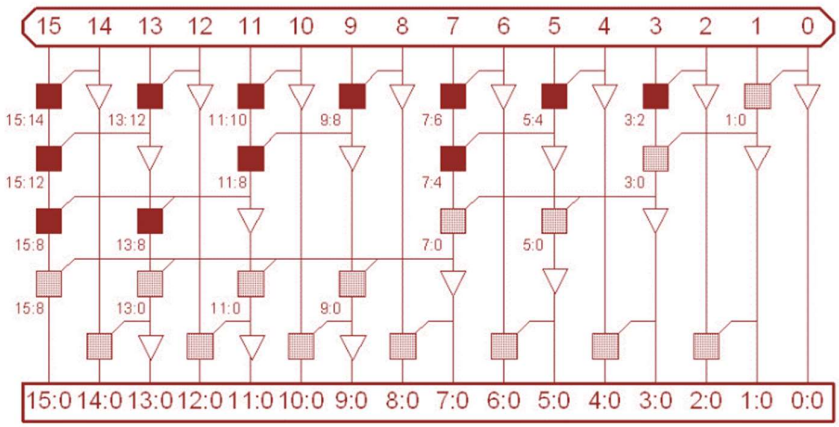

Fig. 2.2.3 Exact 16-Bit Ladner Fischer Adder

In PPA, the carries are generated in parallel and different types of tree structures differ as to how they provide cells in the intermediate stages i.e. mainly in the prefix stage. The cells are a group

of logic gates that tend to take over the area and can optimize the increase in area by reducing operators in stages; as a result, the number of cells is reduced. One of the advantages of the approximate adders or multipliers is that, it can be used to produce inaccurate result which will not affect the computation. Using this advantage for our proposed work, the cells are limited in the prefix stage of the adder. In place of a single black cell involving group propagate/ generate for the last stage, and can make use of the same black cell and given that the result remains unaffected which is depicted in Fig. 2.2.3 The decision of increasing from the last stage to two or more depends upon the bit size of the architecture. Here in this design, each black cell at the top is used to compute the last two stages. change can be implemented in the design by altering the VHDL module for simulation.

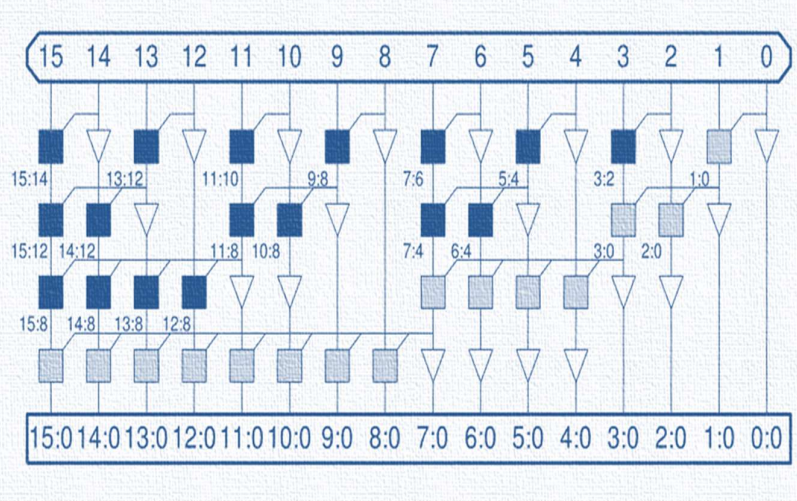

Fig.2.2.4 Exact 16-Bit Sklansky Adder Structure

A Parallel prefix adder that can add binary integers quickly is the Sklansky adder Fig.2.2.4. There are several levels of half adders and complete adders among the input bits. The create and propagate bits for every pair of bits are calculated at each level. The carry bits for the following level are then determined using these bits. The sum bits and the carry-out bit are produced at the last level. Because of its logarithmic depth, the Sklansky adder can add n bits in log2 n steps.

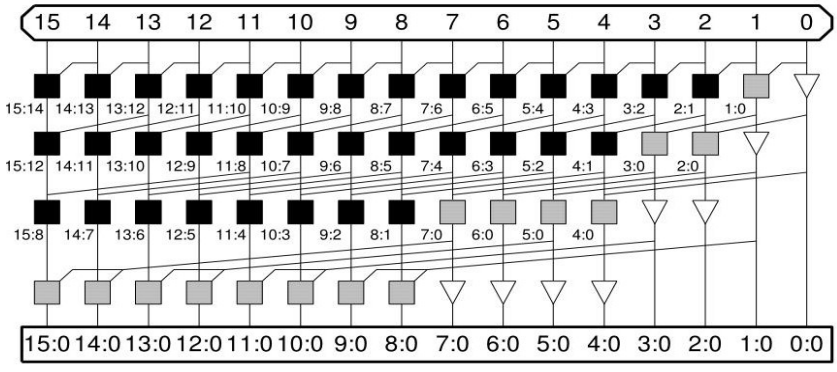

Fig.2.2.5 Exact 16-Bit Knowles Adder Structure

The Fig.2.2.5 Exact 16-Bit Knowles Adder Structure parallel-prefix tree adders are more favourable in terms of speed due to thecomplexity O(log2N) delay (Kogge stone) through the carry path compared to that of other adders. This is similar to kogge stone adder except in Last stage wiring complexity is reduced to half. But it doubles the Loading. Kogge-Stone adder design is the most straightforward, and also it has one of the shortest critical paths of all tree adders. The drawback with the Kogge-Stone adder implementation is the large area consumption and the more complex routing (Fan-Out) of interconnects. Black boxes are BC and grey cells are GC that cannot be removed, because they are required for carry propagation and other calculations.

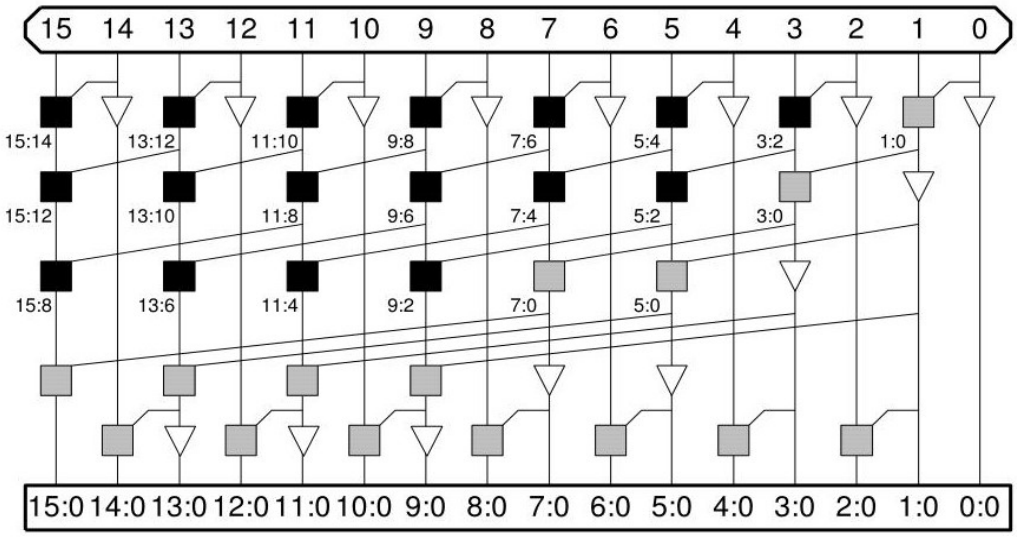

Fig. 2.2.6 Exact 16-Bit Han-Carlson Adder Structure

Han-Carlson adder contains a good trade-off between fan out, number of logic levels and number of black cells. Because of this, Han-Carlson adder can achieve equal to speed execution admiration to Kogge-Stone adder, at lower power utilization and territory. In this manner it is fascinating to execute a speculative Han-Carlson adder. Moved by these reasons, we have generated a Han-Carlson speculative prefix-processing stage by removing the final rows of the Kogge-Stone part of the adder. As an example, Figure 2.2.6 show the Han-Carlson adder in which the two Brent-Kung rows at the initial and toward the end of the graph are unaltered, while the last Kogge Stone row is pruned. This yields a speculative stage with K=8=n/2.in general, one has K ==n/2p. where P is the quantity of pruned levels; the number of levels of the speculative Han-Carlson stage lessens from 1+ log(n) to1+ log(k) (assuming that is a power of two).

CHAPTER 3

PROPOSED APPROXIMATE PPA

3.1 INTRODUCTION

Proposed Approximate parallel prefix adders represent a fascinating intersection of two important concepts in digital circuit design: parallel prefix adders and approximate computing. Traditional parallel prefix adders, as mentioned earlier, excel in high-speed arithmetic applications by breaking down addition into parallel stages. On the other hand, approximate computing techniques aim to trade off accuracy for gains in power, area, or speed. These adders are designed to produce approximate results with acceptable accuracy, often sacrificing precision in favor of efficiency. The motivation behind Proposed approximate parallel prefix adders lies in applications where perfect accuracy isn't crucial or where the computational cost of achieving perfect accuracy outweighs the benefits. Examples include multimedia processing, machine learning inference, and certain types of signal processing. Designing proposed approximate parallel prefix adders involves careful consideration of trade-offs between accuracy, speed, power consumption, and area utilization.

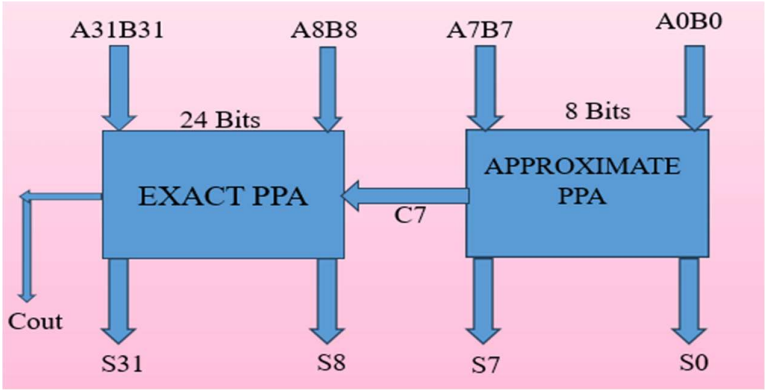

Fig. 3.1 Block diagram of Proposed Approximate PPA

The Proposed approximate PPA, shown in Figure 3.1, is a 32-bit adder that is divided into two phases. It comprises of one distinct Exact PPA and one Approximate Structure. The approximate 8-bit structure, which takes bits from LSB 0 to 7, of inputs A and B, produces approximate sum output vales from S_0 to S_7 using the logical OR operation in the final stage and carry output C_7. This carry output C_7 is used as carry input for the next stage of exact 24-bit adder. The Exact 24-bit adder structure, taking bits from 8 to 31 of inputs A and B, generates a sum output with Exact values from S_8 to S_{31} and carry out of C_{out}.

3.2 PROPOSED APPROXIMATE BRENT KUNG PPA

The Proposed Hybrid Approximate Brent Kung PPA, shown in Figure 3.2, is a 32-bit adder that is divided into two phases. It comprises of one distinct Exact Brent Kung PPA and one Approximate Structure. The approximate 8-bit structure, which takes bits from LSB 0 to 7, of inputs A and B, produces approximate sum output values from S_0 to S_7 using the logical OR operation in the final stage and carry output C_7. This carry output C_7 is used as carry input for the next stage of exact Brent Kung 24-bits PPA. The Exact Brent Kung 24-bit PPA structure, taking bits from 8 to 31 of inputs A and B, generates a sum output with Exact values from S_8 to S_{31} and carry out of C_{out}.

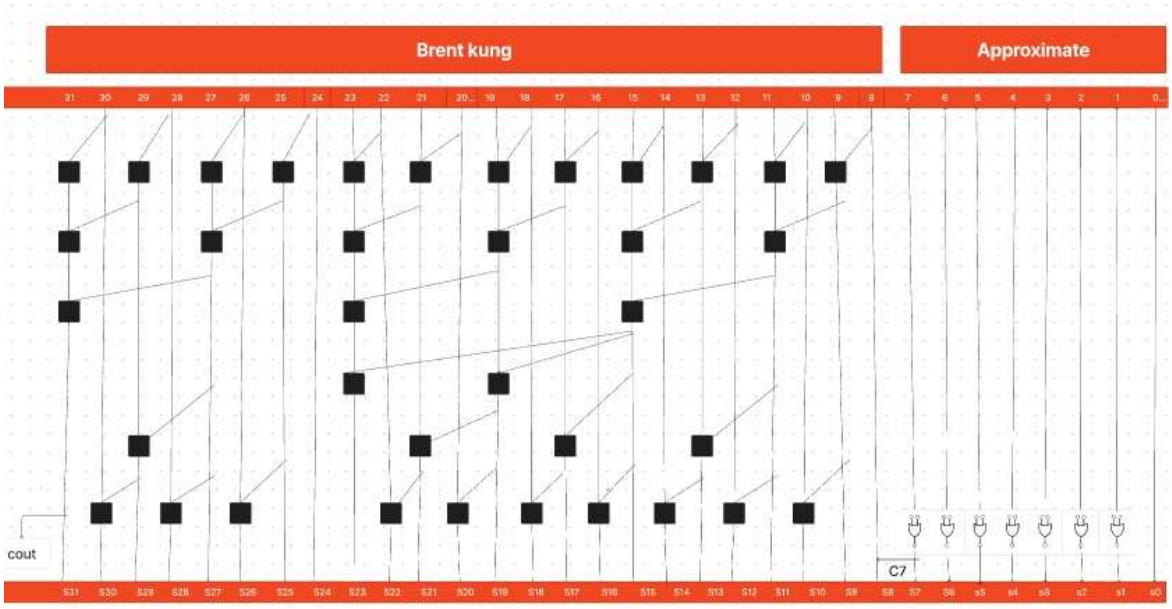

Fig. 3.2 Architecture of Proposed Approximate Brent Kung PPA

3.3 PROPOSED APPROXIMATE KOGGE STONE PPA

The Proposed Hybrid Approximate Kogge Stone PPA, shown in Figure 3.3, is a 32-bit adder that is divided into two phases. It comprises of one distinct Exact Kogge Stone PPA and one Approximate Structure. The approximate 8-bit structure, which takes bits from LSB 0 to 7, of inputs A and B, produces approximate sum output values from S_0 to S_7 using the logical OR operation in the final stage and carry output C_7. This carry output C_7 is used as carry input for the next stage of Exact Kogge Stone 24-bits PPA. The Exact Kogge Stone 24-bits PPA structure, taking bits from 8 to 31 of inputs A and B, generates a sum output with Exact values from S_8 to S_{31} and carry out of C_{out}. The Proposed Approximate Kogge Stone PPA is widely utilized in high-performance computing systems, particularly in 32-bit and 64-bit architectures, as it significantly reduces the critical path.

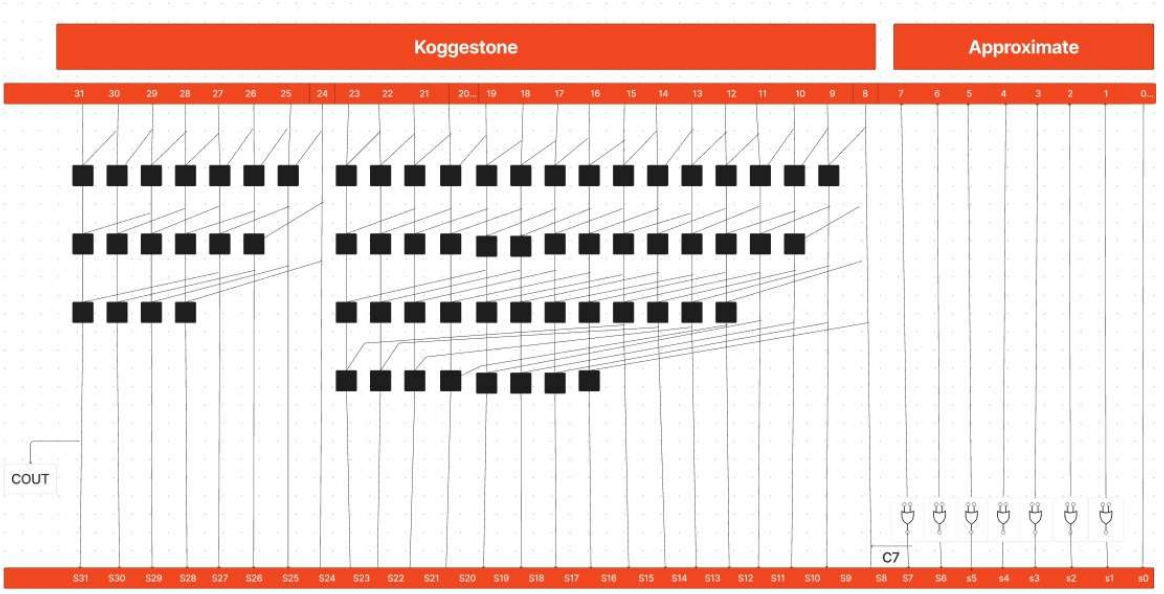

Fig. 3.3 Architecture of Proposed Approximate Kogge Stone PPA

3.4 PROPOSED APPROXIMATE SKLANSKY PPA

The Proposed Hybrid Approximate Sklansky PPA, shown in Figure 3.4, is a 32-bit adder that is divided into two phases. It comprises of one distinct Exact Sklansky PPA and one Approximate Structure. The approximate 8-bit structure, which takes bits from LSB 0 to 7, of inputs A and B, produces approximate sum output values from S_0 to S_7 using the logical OR operation in the final stage and carry output C_7. This carry output C_7 is used as carry input for the next stage of Exact Sklansky 24-bits PPA. The Exact Sklansky 24-bits PPA structure, taking bits from 8 to 31 of inputs A and B, generates a sum output with Exact values from S_8 to S_{31} and carry out of C_{out}.

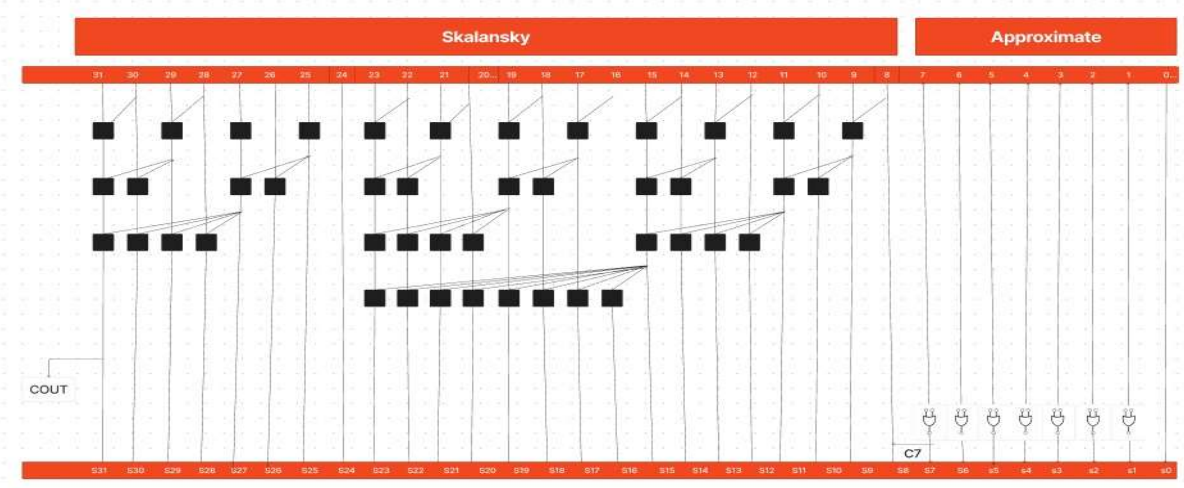

Fig. 3.4 Architecture of Proposed Approximate Sklansky PPA

3.5 PROPOSED APPROXIMATE LADNER-FISCHER PPA

The Proposed Hybrid Approximate Ladner-Fischer PPA, shown in Figure 3.5, is a 32-bit adder that is divided into two phases. It comprises of one distinct Exact Ladner-Fischer PPA and one Approximate Structure. The approximate 8-bit structure, which takes bits from LSB 0 to 7, of inputs A and B, produces approximate sum output values from S_0 to S_7 using the logical OR operation in the final stage and carry output C_7. This carry output C_7 is used as carry input for the next stage of Exact Ladner-Fischer 24-bits PPA. The Exact Ladner-Fischer 24-bits PPA structure, taking bits from 8 to 31 of inputs A and B, generates a sum output with Exact values from S_8 to S_{31} and carry out of C_{out}.

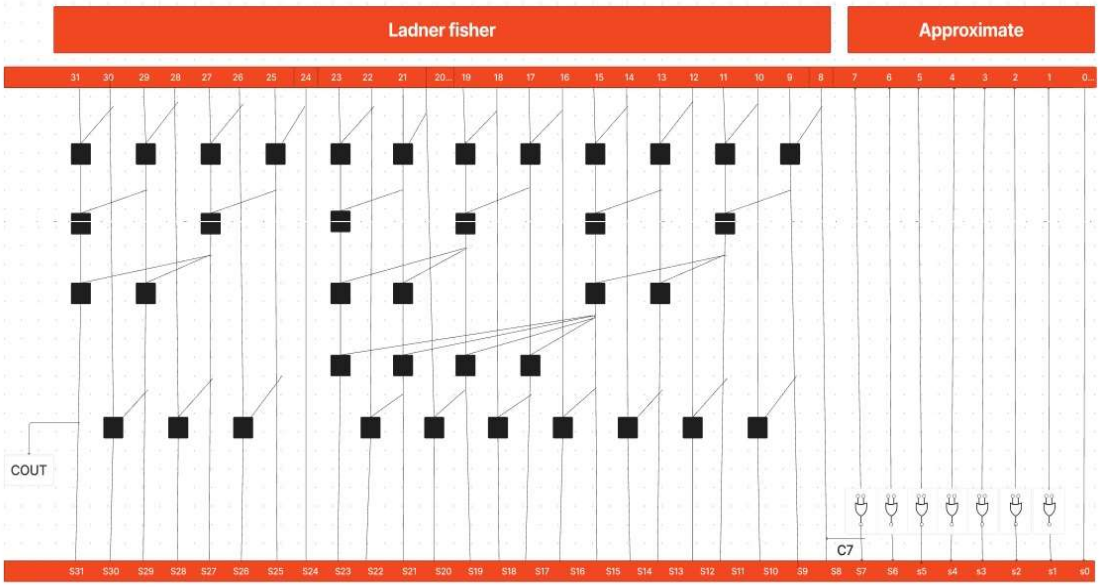

Fig. 3.5 Architecture of Proposed Approximate Ladner-Fischer PPA

3.6 PROPOSED APPROXIMATE KNOWLES PPA

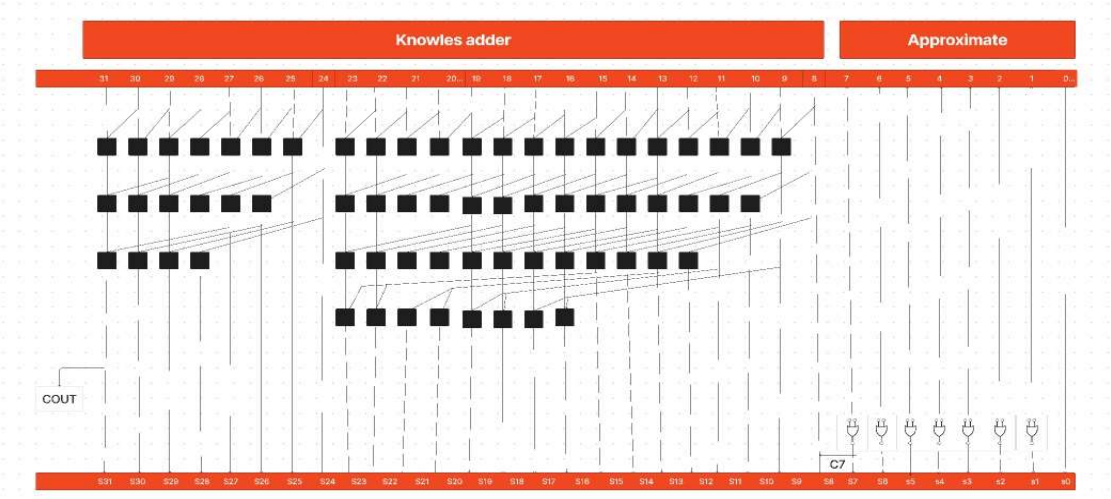

Fig. 3.6 Architecture of Proposed Approximate Knowles PPA

The Proposed Hybrid Approximate Knowles PPA, shown in Figure 3.5, is a 32-bit adder that is divided into two phases. It comprises of one distinct Exact Knowles PPA and one Approximate Structure. The approximate 8-bit structure, which takes bits from LSB 0 to 7, of inputs A and B, produces approximate sum output values from S_0 to S_7 using the logical OR operation in the final stage and carry output C_7. This carry output C_7 is used as carry input for the next stage of Exact 24-bits PPA. The Exact Knowles 24-bits PPA structure, taking bits from 8 to 31 of inputs A and B, generates a sum output with Exact values from S_8 to S_{31} and carry out of C_{out}.

3.7 PROPOSED APPROXIMATE HAN CARLSON PPA

The Proposed Hybrid Approximate Han Carlson PPA, shown in Figure 3.5, is a 32-bit adder that is divided into two phases. It comprises of one distinct Exact Han Carlson PPA and one Approximate Structure. The approximate 8-bit structure, which takes bits from LSB 0 to 7, of inputs A and B, produces approximate sum output values from S_0 to S_7 using the logical OR operation in the final stage and carry output C_7. This carry output C_7 is used as carry input for the next stage of Exact 24-bits PPA. The Exact Han Carlson 24-bits PPA structure, taking bits from 8 to 31 of inputs A and B, generates a sum output with Exact values from S_8 to S_{31} and carry out of C_{out}.

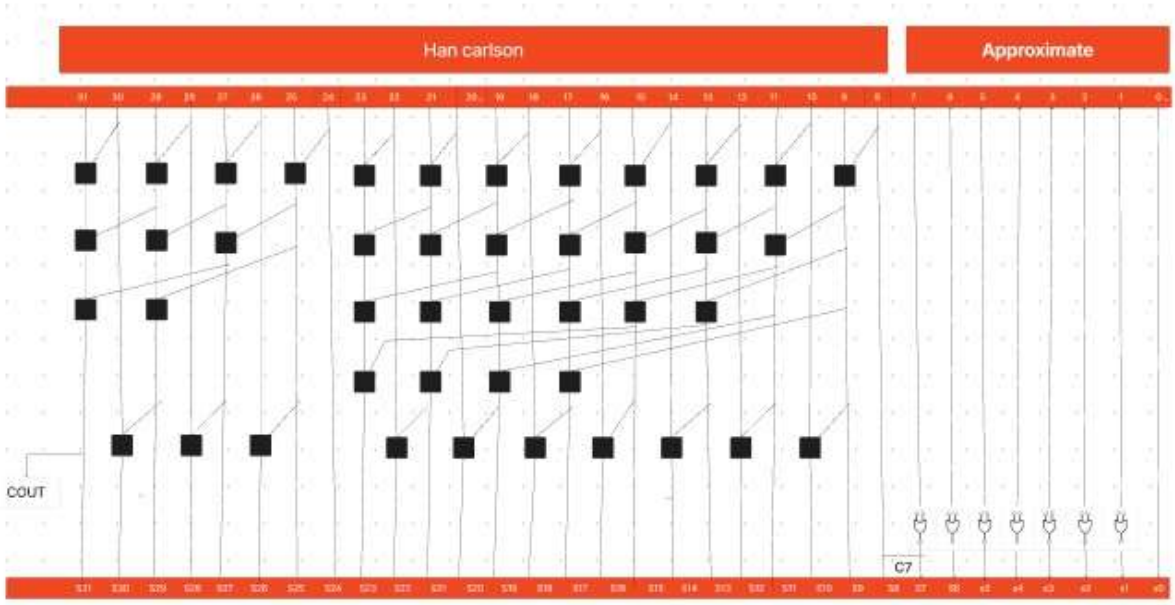

Fig. 3.7 Architecture of Proposed Approximate Han Carlson PPA

CHAPTER 4

PROPOSED HYBRID APPROXIMATE PPA

4.1 INTRODUCTION

A hybrid approximate parallel prefix adder likely refers to a specialized type of circuit used in digital electronics for fast addition of binary numbers. A parallel prefix adder is a type of adder circuit that computes the sum of multiple binary numbers in parallel. It's commonly used in digital systems where high-speed arithmetic operations are required. Approximate computing is a paradigm where computations are performed with acceptable levels of accuracy rather than exact precision. This can lead to faster and more energy-efficient computations at the expense of some loss in accuracy. Combining these two concepts, a hybrid approximate parallel prefix adder likely refers to a specialized adder circuit that incorporates approximate computing techniques within a parallel prefix adder architecture. This could involve using approximate adder cells or approximation techniques within the parallel prefix adder to achieve faster computation speeds or reduce energy consumption, with a trade-off in accuracy. Such a hybrid approach aims to leverage the benefits of both parallelism for speed and approximate computing for efficiency, making it suitable for applications where strict precision is not necessary, such as in machine learning accelerators or multimedia processing units.

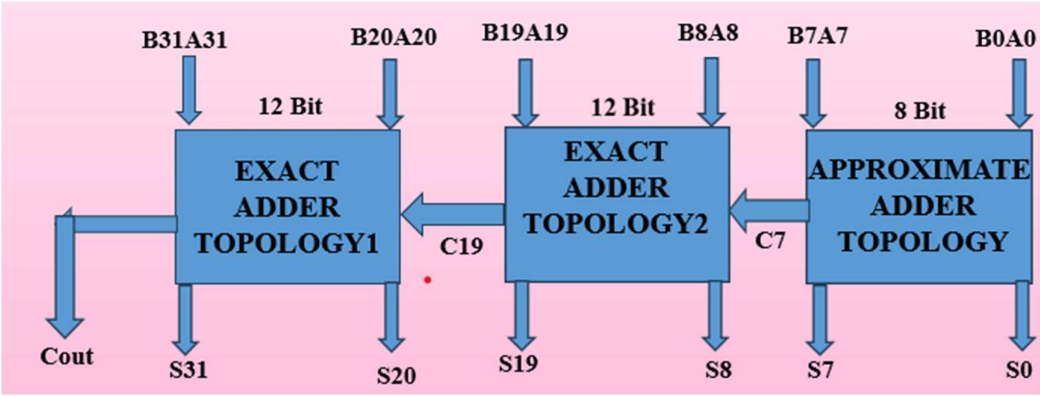

Fig. 4.1.1 Block diagram of Proposed Hybrid Approximate PPA

The Fig. 4.1.1 Block diagram of Proposed Hybrid Approximate PPA represents a hierarchical adder circuit that integrates both exact and approximate adders to achieve a balance between precision and efficiency. The circuit begins with exact adder topology1, which takes two 12-bit inputs ($B_{31}A_{31}$ and $B_{20}A_{20}$) and produces two 12-bit sum outputs (S_{31} and S_{20}) along with a carry output (C_{19}). This carry output (C_{19}) is then fed into the next stage, exact adder topology2, which also processes two 12-bit inputs ($B_{19}A_{19}$ and B_8A_8) along with the carry input from the previous stage, generating two more

12-bit sum outputs (S_{19} and S_8) and another carry output (C_7). Finally, the approximate adder topology takes over, handling two 8-bit inputs (B_7A_7 and B_0A_0) and the carry input from the second exact adder, and producing two 8-bit sum outputs (S_7 and S_0) along with the final carry output (C_{out}). This design leverages the precision of exact adders for the initial 24 bits and uses an approximate adder for the last 8 bits, potentially reducing power consumption and area at the expense of some accuracy.

4.2 PROPOSED HYBRID APPROXIMATE PPA1

The Proposed Hybrid approximate PPA1, shown in Figure 4.2, is a 32-bit adder that is divided into three phases. It comprises of two distinct Exact PPAs and one Approximate Structure. The approximate 8-bit structure, which takes bits from LSB 0 to 7, of inputs A and B, produces approximate sum output values from S_0 to S_7 using the logical OR operation in the final stage and carry output C_7. This carry output C_7 is used as carry input for the next stage of exact Sklansky 12-bit adder. The Sklansky 12-bit adder structure, taking bits from 8 to 19 of inputs A and B, generates a sum output with Exact values from S_8 to S_{19} and carry out of C_{19}. This carry out C_{19} is taken as input for the next exact Ladner Fischer 12-bit Adder. The Ladner Fischer 12-bit adder structure, taking bits from 20 to 31 of inputs A and B, generates a sum output with exact values from S_{20} to S_{31} and carry out of Cout at the end. This Proposed hybrid Approximate PPA1 offers a reduction in area and less delay compared to other Proposed adders.

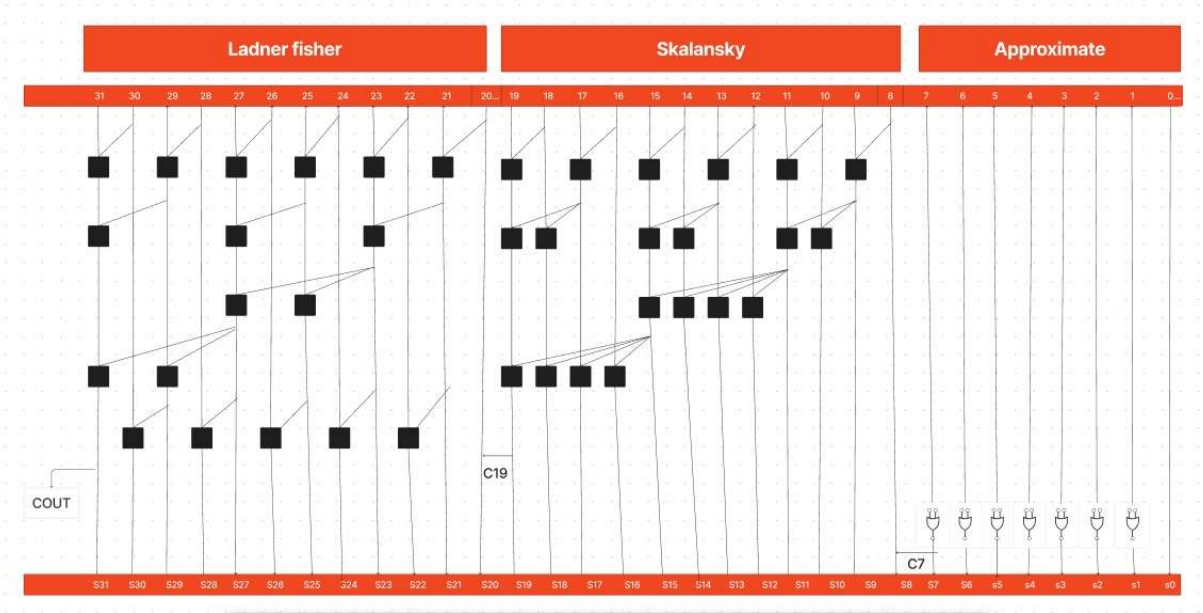

Fig. 4.2 Proposed Hybrid Approximate PPA1

4.3 PROPOSED HYBRID APPROXIMATE PPA2

The Proposed Hybrid approximate PPA2, shown in Figure 4.3, is a 32-bit adder that is divided into three phases. It comprises of two distinct Exact PPAs and one Approximate Structure. The

approximate 8-bit structure, which takes bits from LSB 0 to 7, of inputs A and B, produces approximate sum output values from S_0 to S_7 using the logical OR operation in the final stage and carry output C_7. This carry output C_7 is used as carry input for the next stage of exact Ladner Fischer 12-bit adder. The Ladner Fischer 12-bit adder structure, taking bits from 8 to 19 of inputs A and B, generates a sum output with Exact values from S_8 to S_{19} and carry out of C_{19}. This carry out C_{19} is taken as input for the next exact Sklansky 12-bit Adder. The Sklansky 12-bit adder structure, taking bits from 20 to 31 of inputs A and B, generates a sum output with exact values from S_{20} to S_{31} and carry out of Cout at the end. This Proposed hybrid Approximate PPA2 offers a reduction in area and less delay compared to other Proposed adders.

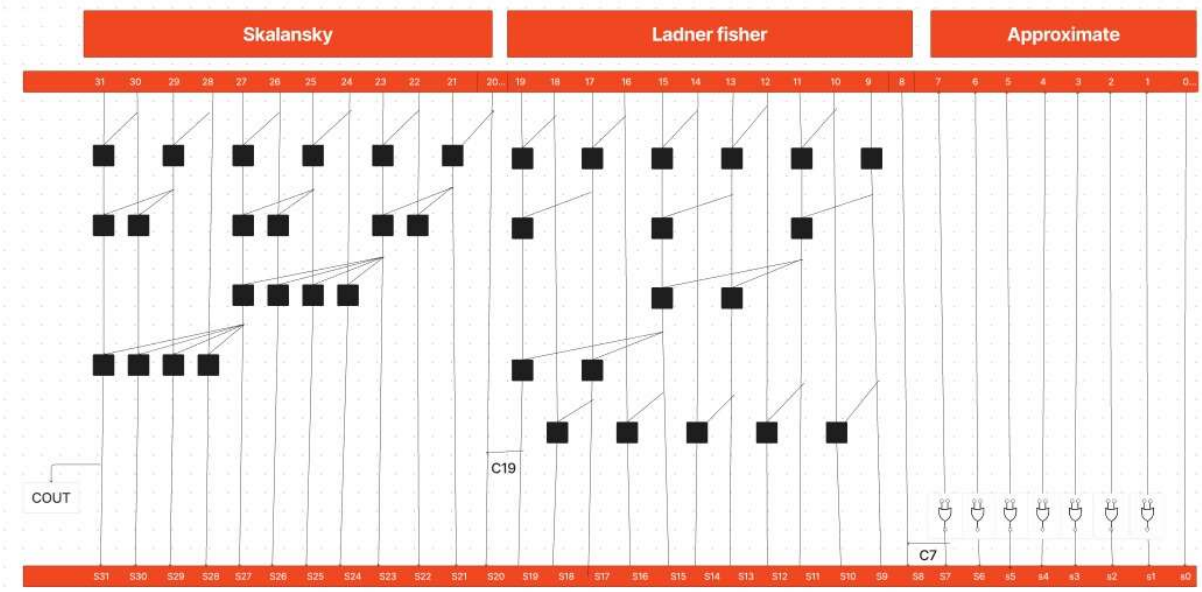

Fig. 4.3 Proposed Hybrid Approximate PPA2

4.4 PROPOSED HYBRID APPROXIMATE PPA3

The Proposed Hybrid approximate PPA3, shown in Figure 4.4, is a 32-bit adder that is divided into three phases. It comprises of two distinct Exact PPAs and one Approximate Structure. The approximate 8-bit structure, which takes bits from LSB 0 to 7, of inputs A and B, produces approximate sum output values from S_0 to S_7 using the logical OR operation in the final stage and carry output C_7. This carry output C_7 is used as carry input for the next stage of exact Knowles 12-bit adder. The Knowles 12-bit adder structure, taking bits from 8 to 19 of inputs A and B, generates a sum output with Exact values from S_8 to S_{19} and carry out of C_{19}. This carry out C_{19} is taken as input for the next exact Sklansky 12-bit Adder. The Sklansky 12-bit adder structure, taking bits from 20 to 31 of inputs A and B, generates a sum output with exact values from S_{20} to S_{31} and carry out of Cout

at the end. This Proposed hybrid Approximate PPA3 offers a reduction in area and less delay compared to other Proposed adders.

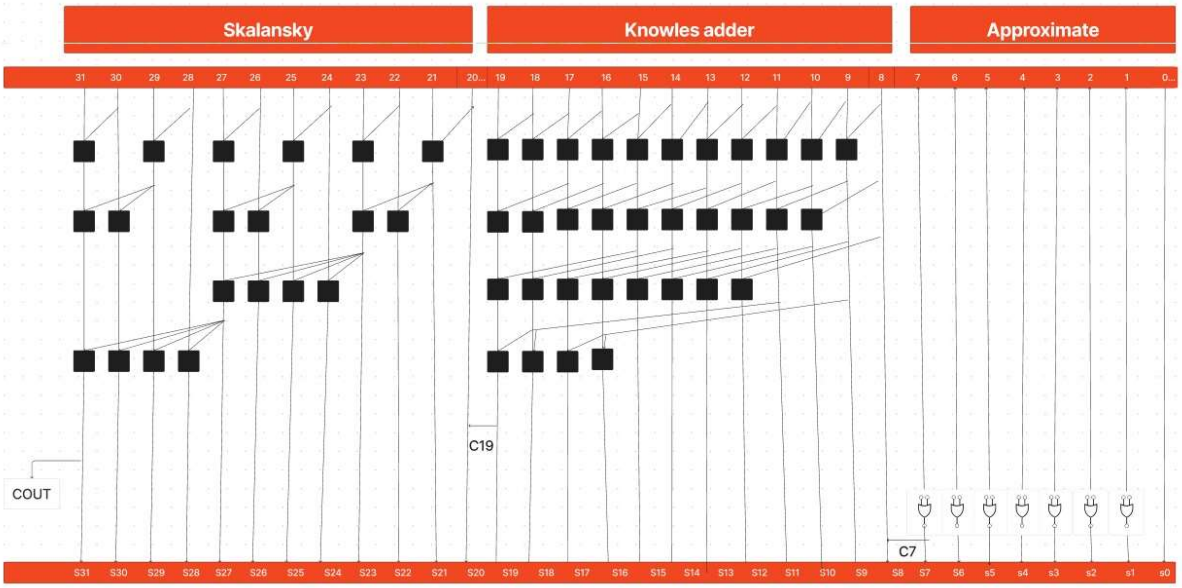

Fig. 4.4 Proposed Hybrid Approximate PPA3

4.5 PROPOSED HYBRID APPROXIMATE PPA4

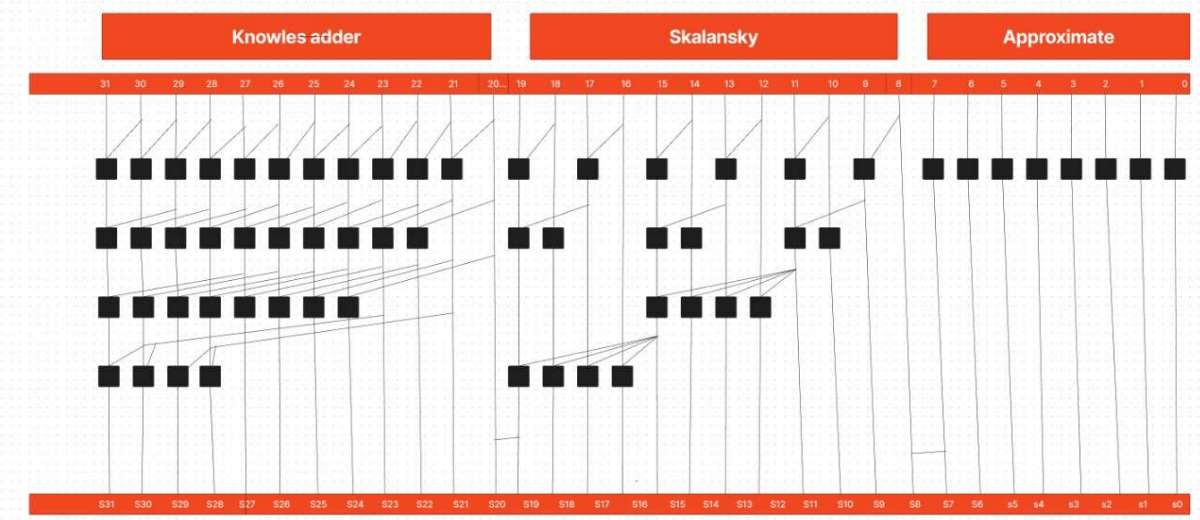

Fig. 4.5 Proposed Hybrid Approximate PPA4

The Proposed Hybrid approximate PPA4, shown in Figure 4.5, is a 32-bit adder that is divided into three phases. It comprises of two distinct Exact PPAs and one Approximate Structure. The approximate 8-bit structure, which takes bits from LSB 0 to 7, of inputs A and B, produces approximate sum output values from S_0 to S_7 using the logical OR operation in the final stage and carry output C_7. This carry output C_7 is used as carry input for the next stage of exact Sklansky 12-bit adder. The Sklansky 12-bit adder structure, taking bits from 8 to 19 of inputs A and B, generates

a sum output with Exact values from S_8 to S_{19} and carry out of C_{19}. This carry out C_{19} is taken as input for the next exact Knowles 12-bit Adder. The Knowles 12-bit adder structure, taking bits from 20 to 31 of inputs A and B, generates a sum output with exact values from S_{20} to S_{31} and carry out of Cout at the end. This Proposed hybrid Approximate PPA4 offers a reduction in area and less delay compared to other Proposed adders.

4.6 PROPOSED HYBRID APPROXIMATE PPA5

The Proposed Hybrid approximate PPA5, shown in Figure 4.6, is a 32-bit adder that is divided into three phases. It comprises of two distinct Exact PPAs and one Approximate Structure. The approximate 8-bit structure, which takes bits from LSB 0 to 7, of inputs A and B, produces approximate sum output values from S_0 to S_7 using the logical OR operation in the final stage and carry output C_7. This carry output C_7 is used as carry input for the next stage of exact Knowles 12-bit adder. The Knowles 12-bit adder structure, taking bits from 8 to 19 of inputs A and B, generates a sum output with Exact values from S_8 to S_{19} and carry out of C_{19}. This carry out C19 is taken as input for the next exact Ladner Fischer 12-bit Adder. The Ladner Fischer 12-bit adder structure, taking bits from 20 to 31 of inputs A and B, generates a sum output with exact values from S_{20} to S_{31} and carry out of Cout at the end. This Proposed hybrid Approximate PPA5 offers a reduction in area and less delay compared to other Proposed adders.

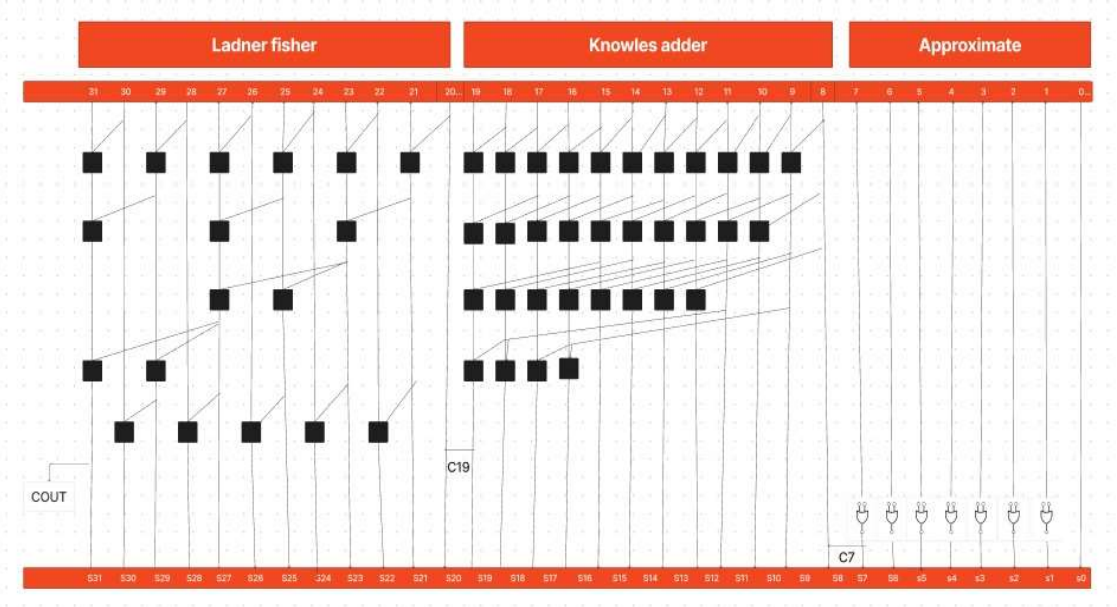

Fig. 4.6 Proposed Hybrid Approximate PPA5

4.7 PROPOSED HYBRID APPROXIMATE PPA6

The Proposed Hybrid approximate PPA6, shown in Figure 4.7, is a 32-bit adder that is divided into three phases. It comprises of two distinct Exact PPAs and one Approximate Structure. The

approximate 8-bit structure, which takes bits from LSB 0 to 7, of inputs A and B, produces approximate sum output values from S_0 to S_7 using the logical OR operation in the final stage and carry output C_7. This carry output C_7 is used as carry input for the next stage of exact Ladner Fischer 12-bit adder. The Ladner Fischer 12-bit adder structure, taking bits from 8 to 19 of inputs A and B, generates a sum output with Exact values from S_8 to S_{19} and carry out of C_{19}. This carry out C_{19} is taken as input for the next exact Knowles 12-bit Adder. The Knowles 12-bit adder structure, taking bits from 20 to 31 of inputs A and B, generates a sum output with exact values from S_{20} to S_{31} and carry out of Cout at the end. This Proposed hybrid Approximate PPA6 offers a reduction in and less delay compared to other Proposed adders.

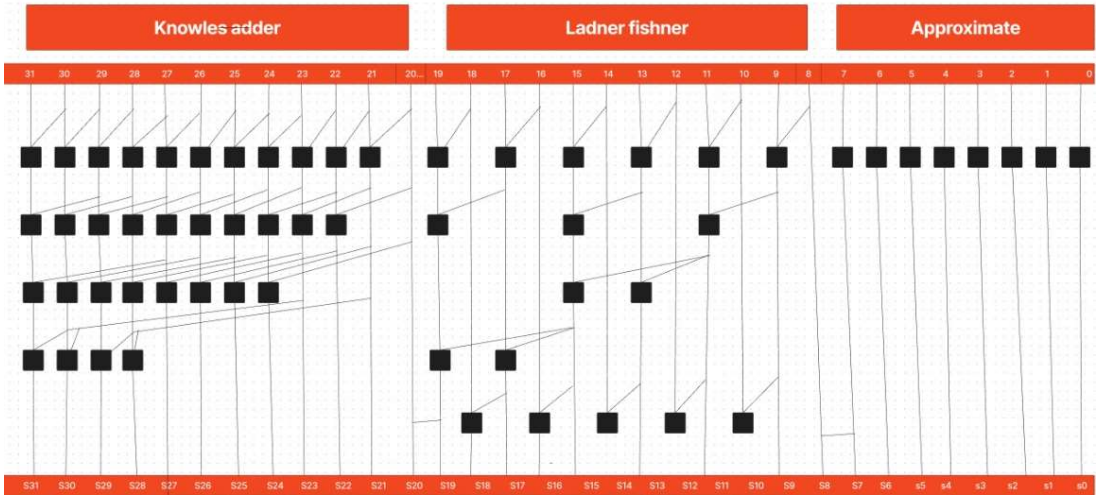

Fig. 4.7 Proposed Hybrid Approximate PPA6

4.8 PROPOSED HYBRID APPROXIMATE PPA7

The Proposed Hybrid approximate PPA7, shown in Figure 4.8, is a 32-bit adder that is divided into three phases. It comprises of two distinct Exact PPAs and one Approximate Structure. The approximate 8-bit structure, which takes bits from LSB 0 to 7, of inputs A and B, produces approximate sum output values from S_0 to S_7 using the logical OR operation in the final stage and carry output C_7. This carry output C_7 is used as carry input for the next stage of exact Brent Kung 12-bit adder. The Brent Kung 12-bit adder structure, taking bits from 8 to 19 of inputs A and B, generates a sum output with Exact values from S_8 to S_{19} and carry out of C_{19}. This carry out C_{19} is taken as input for the next exact Ladner Fischer 12-bit Adder. The Ladner Fischer 12-bit adder structure, taking bits from 20 to 31 of inputs A and B, generates a sum output with exact values from S_{20} to S_{31} and carry out of Cout at the end. This Proposed hybrid Approximate PPA7 offers a reduction in and less delay compared to other Proposed adders.

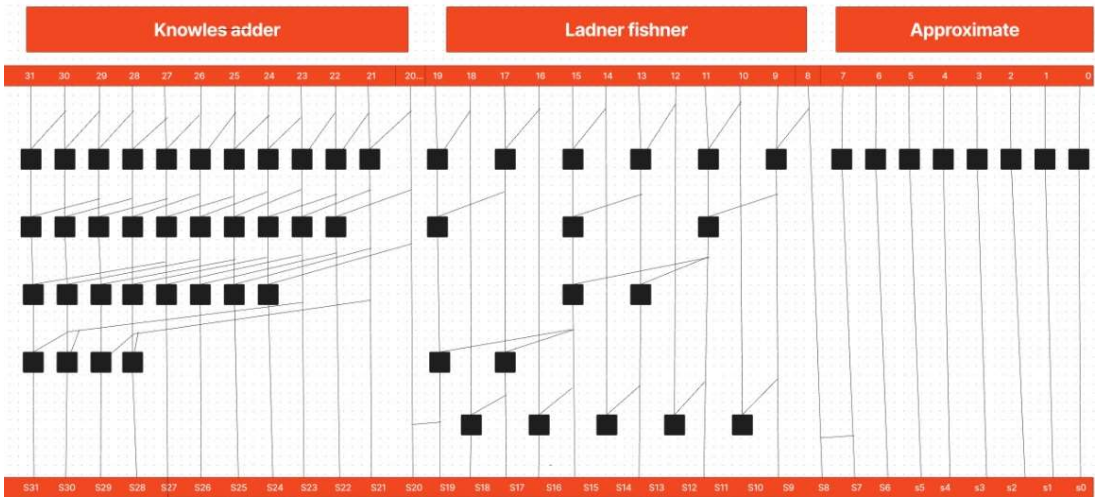

Fig. 4.7 Proposed Hybrid Approximate PPA6

4.8 PROPOSED HYBRID APPROXIMATE PPA7

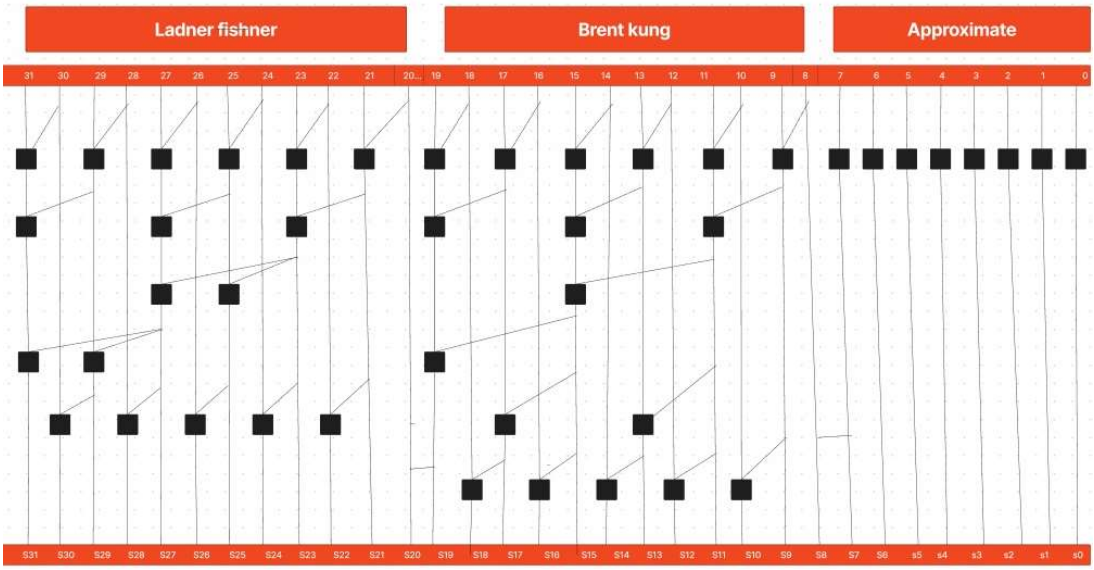

Fig. 4.8 Proposed Hybrid Approximate PPA7

The Proposed Hybrid approximate PPA7, shown in Figure 4.8, is a 32-bit adder that is divided into three phases. It comprises of two distinct Exact PPAs and one Approximate Structure. The approximate 8-bit structure, which takes bits from LSB 0 to 7, of inputs A and B, produces approximate sum output values from S_0 to S_7 using the logical OR operation in the final stage and carry output C_7. This carry output C_7 is used as carry input for the next stage of exact Brent Kung 12-bit adder. The Brent Kung 12-bit adder structure, taking bits from 8 to 19 of inputs A and B, generates a sum output with Exact values from S_8 to S_{19} and carry out of C_{19}. This carry out C_{19} is taken as input for the next exact Ladner Fischer 12-bit Adder. The Ladner Fischer 12-bit adder structure, taking bits from 20 to 31 of inputs A and B, generates a sum output with exact values from S_{20} to S_{31} and carry out of Cout at the end.

4.9 PROPOSED HYBRID APPROXIMATE PPA8

The Proposed Hybrid approximate PPA8, shown in Figure 4.9, is a 32-bit adder that is divided into three phases. It comprises of two distinct Exact PPAs and one Approximate Structure. The approximate 8-bit structure, which takes bits from LSB 0 to 7, of inputs A and B, produces approximate sum output values from S_0 to S_7 using the logical OR operation in the final stage and carry output C_7. This carry output C_7 is used as carry input for the next stage of exact Brent Kung 12-bit adder. The Brent Kung 12-bit adder structure, taking bits from 8 to 19 of inputs A and B, generates a sum output with Exact values from S_8 to S_{19} and carry out of C_{19}. This carry out C_{19} is taken as input for the next exact Sklansky 12-bit Adder. The Sklansky 12-bit adder structure, taking bits from 20 to 31 of inputs A and B, generates a sum output with exact values from S_{20} to S_{31} and carry out of Cout at the end.

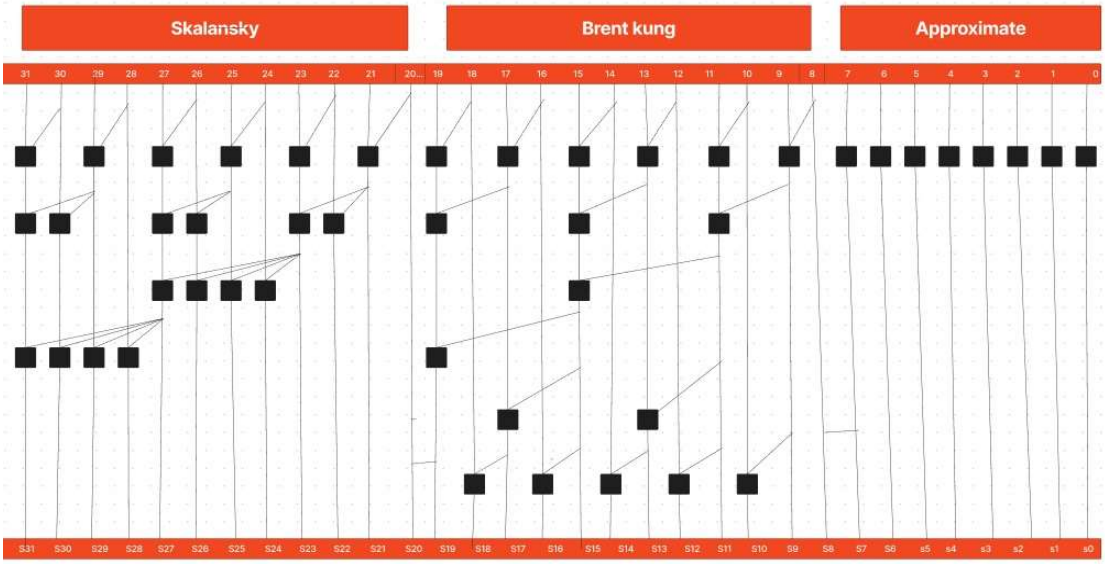

Fig. 4.9 Proposed Hybrid Approximate PPA8

4.10 PROPOSED HYBRID APPROXIMATE PPA9

The Proposed Hybrid approximate PPA9, shown in Figure 4.10, is a 32-bit adder that is divided into three phases. It comprises of two distinct Exact PPAs and one Approximate Structure. The approximate 8-bit structure, which takes bits from LSB 0 to 7, of inputs A and B, produces approximate sum output values from S0 to S7 using the logical OR operation in the final stage and carry output C7. This carry output C7 is used as carry input for the next stage of exact Kogge stone 12-bit adder. The Kogge stone 12-bit adder structure, taking bits from 8 to 19 of inputs A and B, generates a sum output with Exact values from S8 to S19 and carry out of C19. This carry out C19 is taken as input for the next exact Ladner Fischer 12-bit Adder. The Ladner Fischer 12-bit adder structure, taking bits from 20 to 31 of inputs A and B, generates a sum output with exact values from S20 to S31 and carry out of Cout at the end.

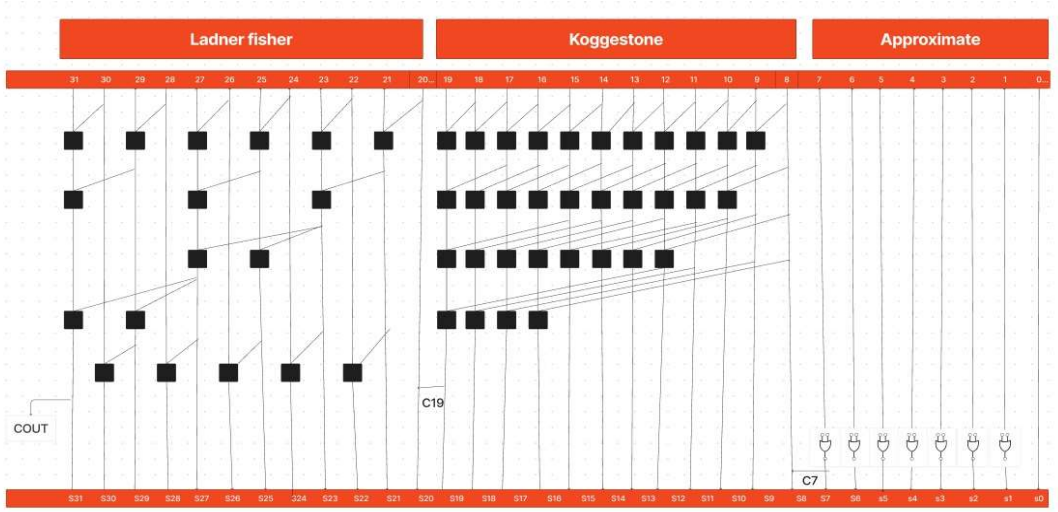

Fig. 4.10 Proposed Hybrid Approximate PPA9

4.11 PROPOSED HYBRID APPROXIMATE PPA10

The Proposed Hybrid approximate PPA10, shown in Figure 4.11, is a 32-bit adder that is divided into three phases. It comprises of two distinct Exact PPAs and one Approximate Structure. The approximate 8-bit structure, which takes bits from LSB 0 to 7, of inputs A and B, produces approximate sum output values from S0 to S7 using the logical OR operation in the final stage and carry output C7. This carry output C7 is used as carry input for the next stage of exact Kogge stone 12-bit adder. The Kogge stone 12-bit adder structure, taking bits from 8 to 19 of inputs A and B, generates a sum output with Exact values from S8 to S19 and carry out of C19. This carry out C19 is taken as input for the next exact Sklansky 12-bit Adder. The Sklansky 12-bit adder structure, taking bits from 20 to 31 of inputs A and B, generates a sum output with exact values from S20 to S31 and carry out of Cout at the end.

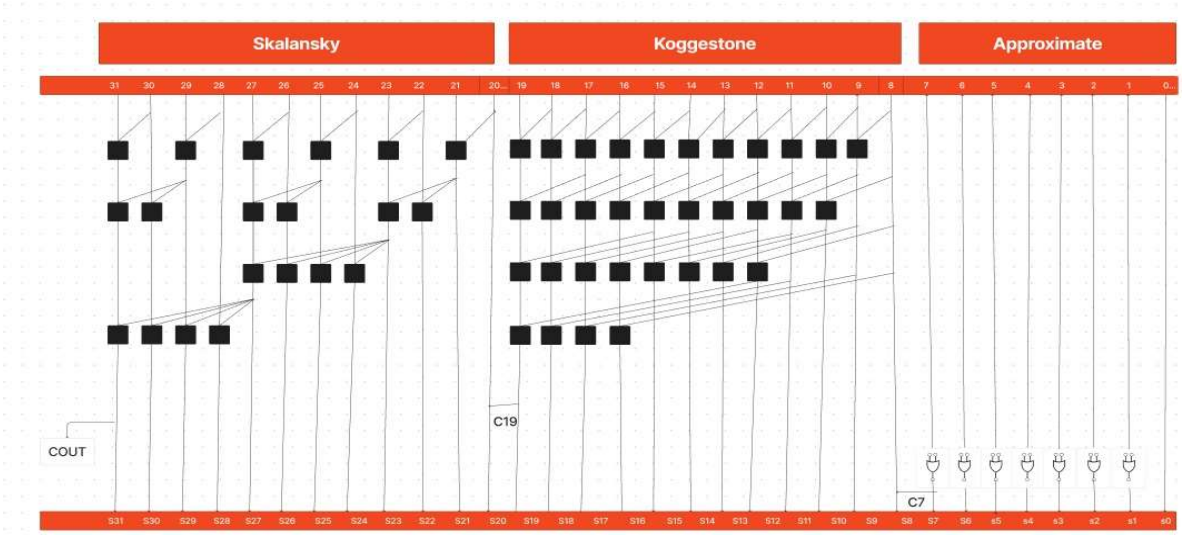

Fig. 4.11 Proposed Hybrid Approximate PPA10

CHAPTER 5

PROPOSED HYBRID APPROXIMATE PPA

5.1 INTRODUCTION

The Proposed Hybrid approximate PPA, shown in Figure.5.1, is a 32-bit adder that is divided into Four phases. It comprises of three distinct Exact PPAs and one Approximate Structure. The approximate 8-bit structure, which takes bits from 0 to 7, of inputs A and B, produces approximate sum output values from S_0 to S_7 and carry output C_7. This carry output C_7 is used as carry input for the next stage of exact 8-bit adder. The Exact 8-bit adder topology1 structure, taking bits from 8 to 15 of inputs A and B, generates a sum output with Exact values from S_8 to S_{15} and carry out of C_{15}. This carry out C_{15} is taken as input for the next exact 8-bit Adder. The exact 8-bit adder topology2 structure, taking bits from 16 to 23 of inputs A and B, generates a sum output with exact values from S16 to S23 and carry out 23. The exact 8-bit adder topology 3 structure, taking bits from 24 to 31 of inputs A and B, generates a sum output with Exact values from S_{24} to S_{31} and carry out of C_{out}.

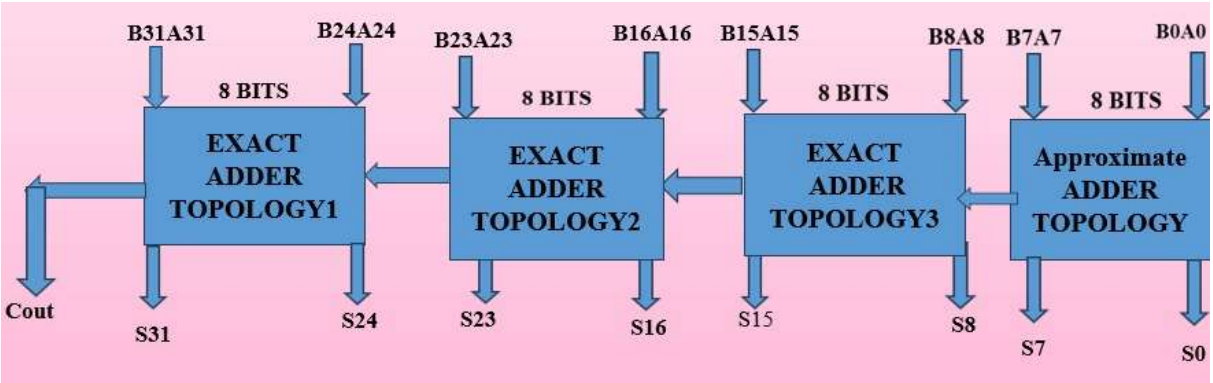

Fig. 5.1.1 Block diagram of Hybrid Approximate PPA

5.2 PROPOSED HYBRID APPROXIMATE PPA11

The Proposed Hybrid approximate PPA11, shown in Figure 5.2, is a 32-bit adder that is divided into Four phases. It comprises of three distinct Exact PPAs and one Approximate Structure. The approximate 8-bit structure, which takes bits from LSB 0 to 7, of inputs A and B, produces approximate sum output vales from S_0 to S_7 using the logical OR operation in the final stage and carry output C_7. This carry output C_7 is used as carry input for the next stage of exact Ladner-Fischer 8-bit adder. The Ladner-Fischer 8-bit adder structure, taking bits from 8 to 15 of inputs A and B, generates a sum output with Exact values from S_8 to S_{15} and carry out of C_{15}. This carry out C_{15} is taken as input for the next exact Knowles 8-bit Adder. The Knowles 8-bit adder structure, taking bits from 16 to 23 of inputs A and B, generates a sum output with exact values from S_{16} to S_{23} and carry out 23. The

Sklansky 8-bit adder structure, taking bits from 24 to 31 of inputs A and B, generates a sum output with Exact values from S_{24} to S_{31} and carry out of C_{out}.

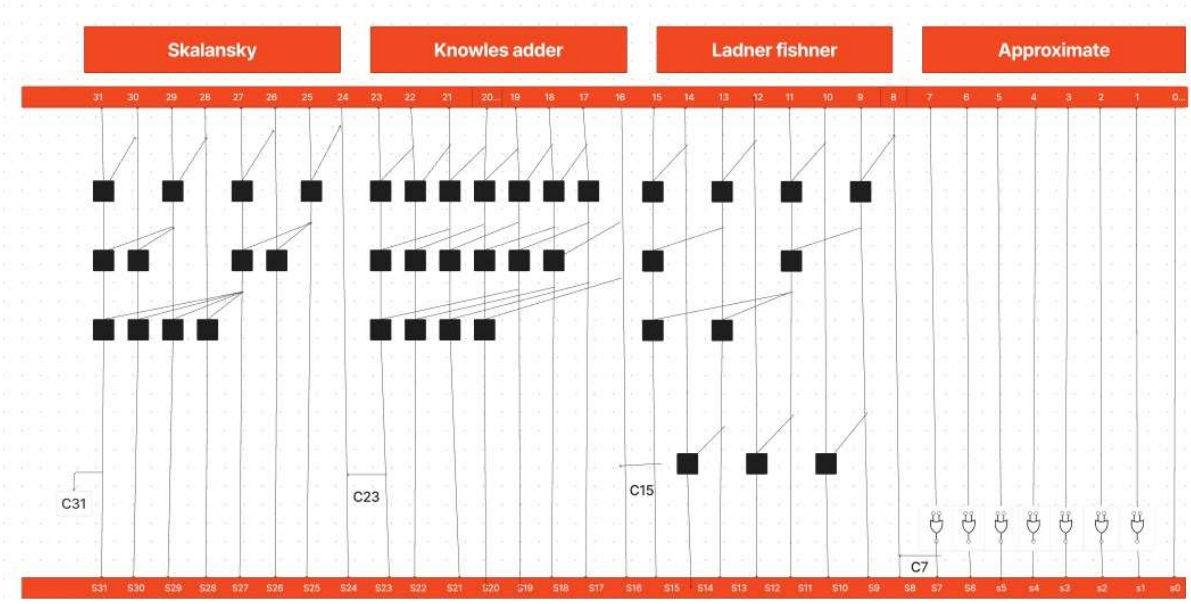

Fig. 5.2 Proposed Hybrid Approximate PPA11

5.3 PROPOSED HYBRID APPROXIMATE PPA12

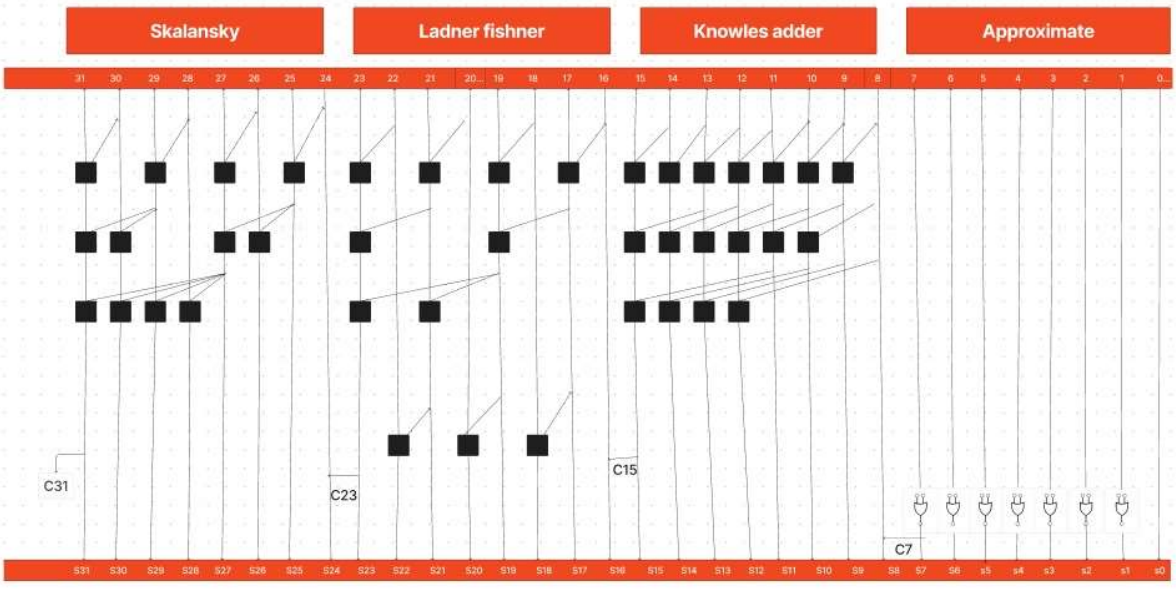

Fig. 5.3 Proposed Hybrid Approximate PPA12

The Proposed Hybrid approximate PPA12, shown in Figure 5.3, is a 32-bit adder that is divided into Four phases. It comprises of three distinct Exact PPAs and one Approximate Structure. The approximate 8-bit structure, which takes bits from LSB 0 to 7, of inputs A and B, produces approximate sum output vales from S_0 to S_7 using the logical OR operation in the final stage and carry output C_7. This carry output C_7 is used as carry input for the next stage of exact Knowles 8-bit adder.

The Knowles 8-bit adder structure, taking bits from 8 to 15 of inputs A and B, generates a sum output with Exact values from S_8 to S_{15} and carry out of C_{15}. This carry out C_{15} is taken as input for the next exact Ladner-Fischer 8-bit Adder. The Ladner-Fischer 8-bit adder structure, taking bits from 16 to 23 of inputs A and B, generates a sum output with exact values from S_{16} to S_{23} and carry out 23. The Sklansky 8-bit adder structure, taking bits from 24 to 31 of inputs A and B, generates a sum output with Exact values from S_{24} to S_{31} and carry out of C_{out}.

5.4 PROPOSED HYBRID APPROXIMATE PPA13

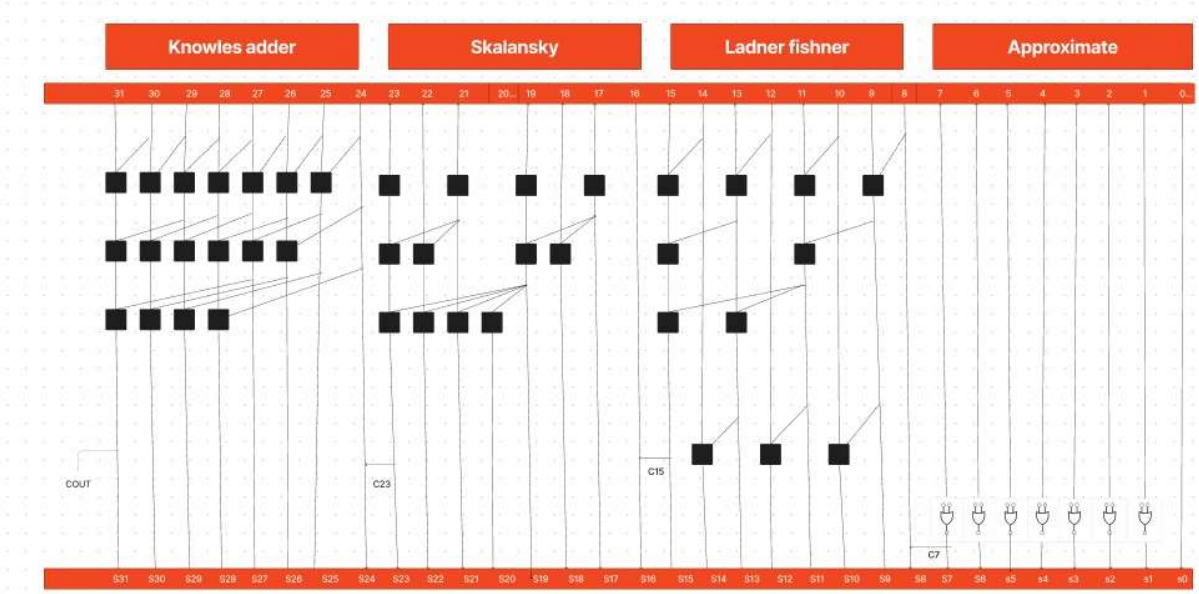

Fig. 5.4 Proposed Hybrid Approximate PPA13

The Proposed Hybrid approximate PPA13, shown in Figure 5.4, is a 32-bit adder that is divided into Four phases. It comprises of three distinct Exact PPAs and one Approximate Structure. The approximate 8-bit structure, which takes bits from LSB 0 to 7, of inputs A and B, produces approximate sum output vales from S_0 to S_7 using the logical OR operation in the final stage and carry output C_7. This carry output C_7 is used as carry input for the next stage of exact Ladner-Fischer 8-bit adder. The Ladner-Fischer 8-bit adder structure, taking bits from 8 to 15 of inputs A and B, generates a sum output with Exact values from S_8 to S_{15} and carry out of C_{15}. This carry out C_{15} is taken as input for the next exact Sklansky 8-bit Adder. The Sklansky 8-bit adder structure, taking bits from 16 to 23 of inputs A and B, generates a sum output with exact values from S_{16} to S_{23} and carry out 23. The Knowles 8-bit adder structure, taking bits from 24 to 31 of inputs A and B, generates a sum output with Exact values from S_{24} to S_{31} and carry out of C_{out}.

5.5 PROPOSED HYBRID APPROXIMATE PPA14

The Proposed Hybrid approximate PPA14, shown in Figure 5.5, is a 32-bit adder that is divided into Four phases. It comprises of three distinct Exact PPAs and one Approximate Structure. The approximate 8-bit structure, which takes bits from LSB 0 to 7, of inputs A and B, produces approximate sum output vales from S_0 to S_7 using the logical OR operation in the final stage and carry output C_7. This carry output C_7 is used as carry input for the next stage of exact Sklansky 8-bit adder. The Sklansky 8-bit adder structure, taking bits from 8 to 15 of inputs A and B, generates a sum output with Exact values from S_8 to S_{15} and carry out of C_{15}. This carry out C_{15} is taken as input for the next exact Ladner Fishner 8-bit Adder. The Ladner Fishner 8-bit adder structure, taking bits from 16 to 23 of inputs A and B, generates a sum output with exact values from S_{16} to S_{23} and carry out 23. The Knowles 8-bit adder structure, taking bits from 24 to 31 of inputs A and B, generates a sum output with Exact values from S_{24} to S_{31} and carry out of C_{out}.

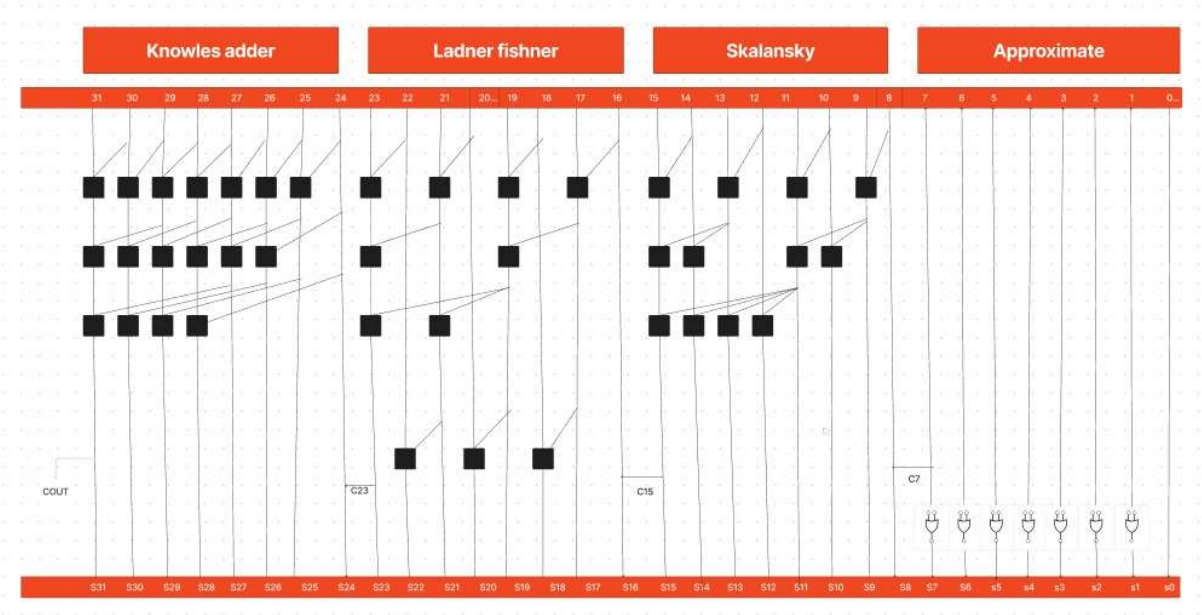

Fig. 5.5 Proposed Hybrid Approximate PPA14

5.6 PROPOSED HYBRID APPROXIMATE PPA15

The Proposed Hybrid approximate PPA15, shown in Figure 5.6, is a 32-bit adder that is divided into Four phases. It comprises of three distinct Exact PPAs and one Approximate Structure. The approximate 8-bit structure, which takes bits from LSB 0 to 7, of inputs A and B, produces approximate sum output vales from S_0 to S_7 using the logical OR operation in the final stage and carry output C_7. This carry output C_7 is used as carry input for the next stage of exact Sklansky 8-bit adder. The Sklansky 8-bit adder structure, taking bits from 8 to 15 of inputs A and B, generates a sum output with Exact values from S_8 to S_{15} and carry out of C_{15}. This carry out C_{15} is taken as input for the next exact Knowles 8-bit Adder. The Knowles 8-bit adder structure, taking bits from 16 to 23 of inputs A and B, generates a sum output with exact values from S_{16} to S_{23} and carry out 23. The Ladner-Fischer

8-bit adder structure, taking bits from 24 to 31 of inputs A and B, generates a sum output with Exact values from S_{24} to S_{31} and carry out of C_{out}.

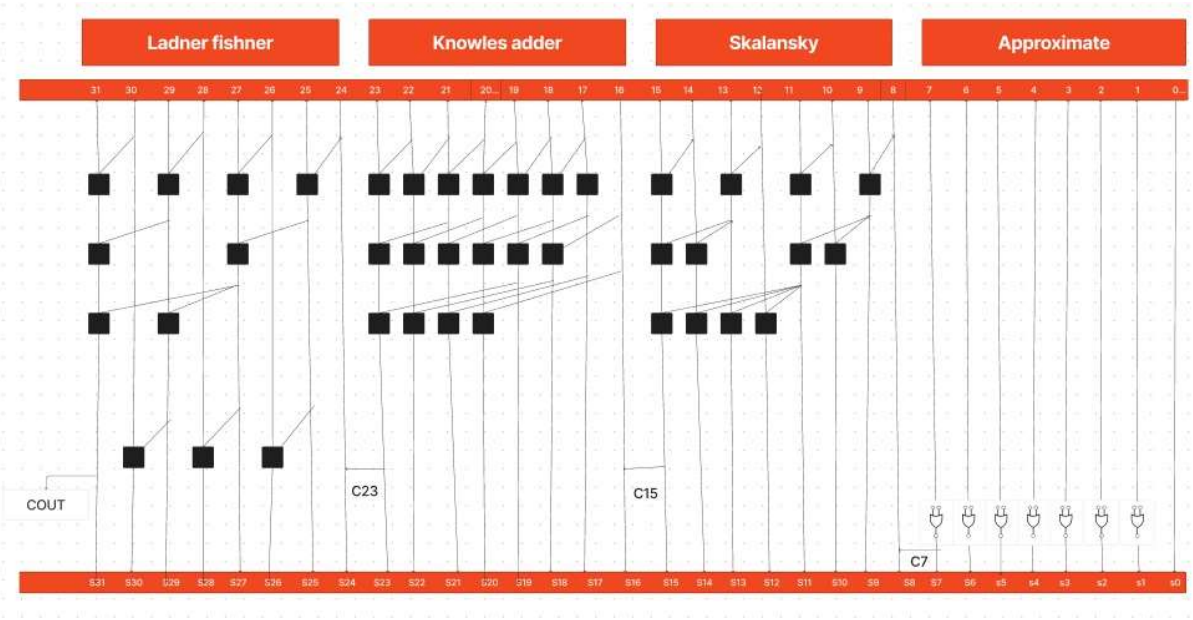

Fig. 5.6 Proposed Hybrid Approximate PPA15

5.7 PROPOSED HYBRID APPROXIMATE PPA16

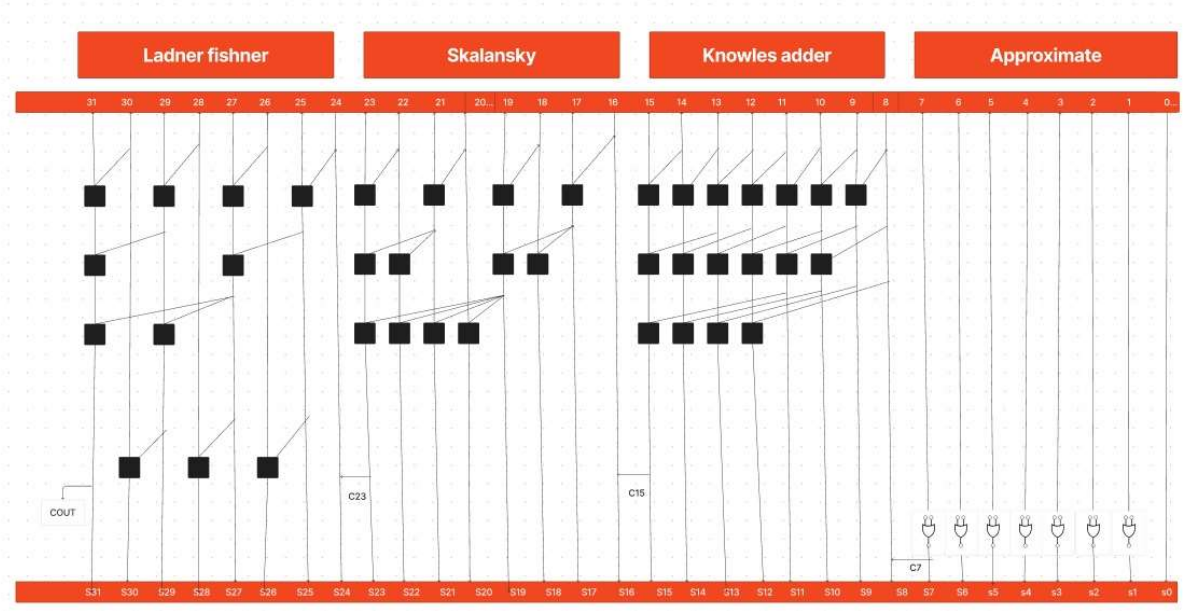

Fig. 5.7 Proposed Hybrid Approximate PPA16

The Proposed Hybrid approximate PPA16, shown in Figure 5.7, is a 32-bit adder that is divided into Four phases. It comprises of three distinct Exact PPAs and one Approximate Structure. The approximate 8-bit structure, which takes bits from LSB 0 to 7, of inputs A and B, produces approximate sum output vales from S_0 to S_7 using the logical OR operation in the final stage and carry

output C_7. This carry output C_7 is used as carry input for the next stage of exact Ladner-Fischer 8-bit adder. The Ladner-Fischer 8-bit adder structure, taking bits from 8 to 15 of inputs A and B, generates a sum output with Exact values from S_8 to S_{15} and carry out of C_{15}. This carry out C_{15} is taken as input for the next exact Sklansky 8-bit Adder. The Sklansky 8-bit adder structure, taking bits from 16 to 23 of inputs A and B, generates a sum output with exact values from S_{16} to S_{23} and carry out 23. The Knowles 8-bit adder structure, taking bits from 24 to 31 of inputs A and B, generates a sum output with Exact values from S_{24} to S_{31} and carry out of C_{out}.

5.8 PROPOSED HYBRID APPROXIMATE PPA17

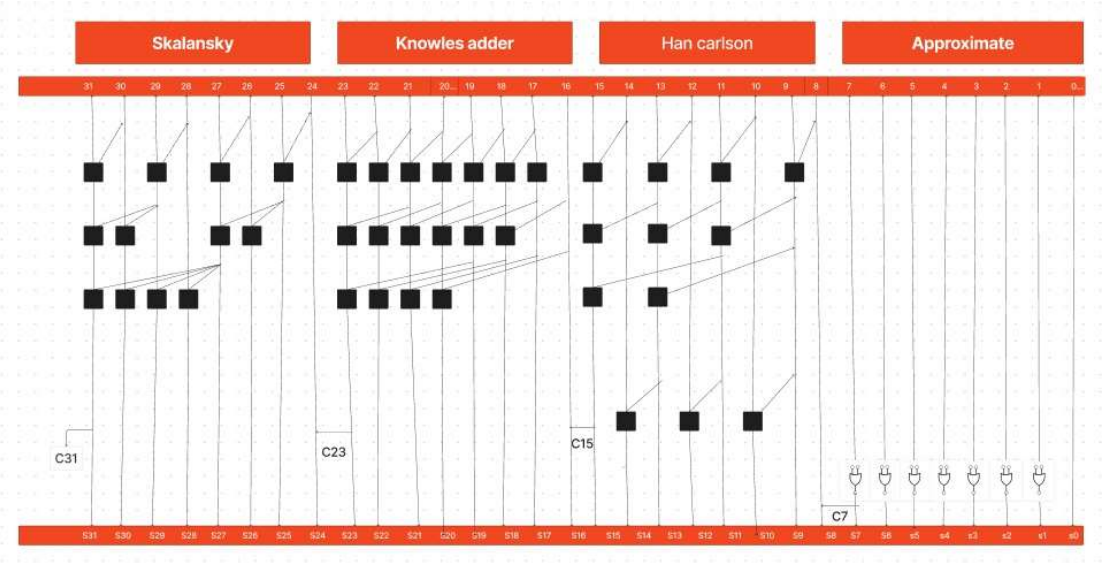

Fig. 5.8 Proposed Hybrid Approximate PPA17

The Proposed Hybrid approximate PPA17, shown in Figure 5.8, is a 32-bit adder that is divided into Four phases. It comprises of three distinct Exact PPAs and one Approximate Structure. The approximate 8-bit structure, which takes bits from LSB 0 to 7, of inputs A and B, produces approximate sum output vales from S_0 to S_7 using the logical OR operation in the final stage and carry output C_7. This carry output C_7 is used as carry input for the next stage of exact Han-Carlson 8-bit adder. The Han-Carlson 8-bit adder structure, taking bits from 8 to 15 of inputs A and B, generates a sum output with Exact values from S_8 to S_{15} and carry out of C_{15}. This carry out C_{15} is taken as input for the next exact Knowles 8-bit Adder. The Knowles 8-bit adder structure, taking bits from 16 to 23 of inputs A and B, generates a sum output with exact values from S_{16} to S_{23} and carry out 23. The Sklansky 8-bit adder structure, taking bits from 24 to 31 of inputs A and B, generates a sum output with Exact values from S_{24} to S_{31} and carry out of C_{out}.

5.9 PROPOSED HYBRID APPROXIMATE PPA18

The Proposed Hybrid approximate PPA18, shown in Figure-5.1, is a 32-bit adder that is divided into Four phases. It comprises of three distinct Exact PPAs and one Approximate Structure. The approximate 8-bit structure, which takes bits from LSB 0 to 7, of inputs A and B, produces approximate sum output vales from S_0 to S_7 using the logical OR operation in the final stage and carry output C_7. This carry output C_7 is used as carry input for the next stage of exact Kogge-Stone 8-bit adder. The Kogge-Stone 8-bit adder structure, taking bits from 8 to 15 of inputs A and B, generates a sum output with Exact values from S_8 to S_{15} and carry out of C_{15}. This carry out C_{15} is taken as input for the next exact Ladner Fishner 8-bit Adder. The Ladner Fishner 8-bit adder structure, taking bits from 16 to 23 of inputs A and B, generates a sum output with exact values from S16 to S23 and carry out 23. The Sklansky 8-bit adder structure, taking bits from 24 to 31 of inputs A and B, generates a sum output with Exact values from S_{24} to S_{31} and carry out of C_{out}.

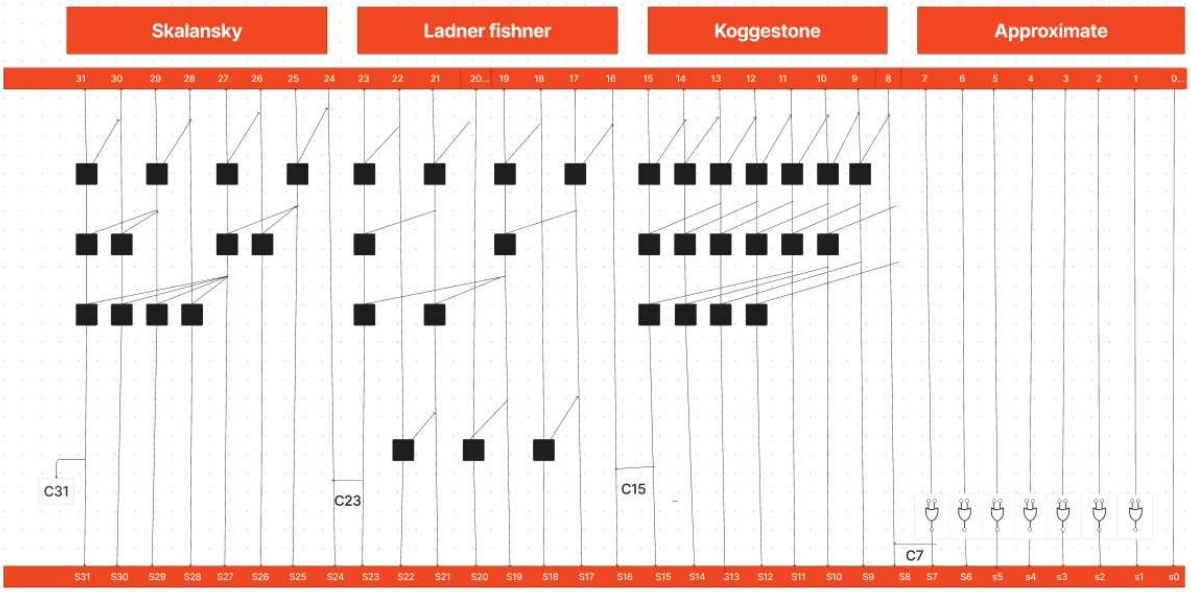

Fig. 5.9 Proposed Hybrid Approximate PPA18

5.10 PROPOSED HYBRID APPROXIMATE PPA19

The Proposed Hybrid approximate PPA19, shown in Figure 5.10, is a 32-bit adder that is divided into Four phases. It comprises of three distinct Exact PPAs and one Approximate Structure. The approximate 8-bit structure, which takes bits from LSB 0 to 7, of inputs A and B, produces approximate sum output vales from S_0 to S_7 using the logical OR operation in the final stage and carry output C_7. This carry output C_7 is used as carry input for the next stage of exact Brent-Kung 8-bit adder. The Brent-Kung 8-bit adder structure, taking bits from 8 to 15 of inputs A and B, generates a sum output with Exact values from S_8 to S_{15} and carry out of C_{15}. This carry out C_{15} is taken as input for the next exact Ladner-Fischer 8-bit Adder. The Ladner-Fischer 8-bit adder structure, taking bits from 16 to 23 of inputs A and B, generates a sum output with exact values from S_{16} to S_{23} and carry

out C_{23}. The Sklansky 8-bit adder structure, taking bits from 24 to 31 of inputs A and B, generates a sum output with Exact values from S_{24} to S_{31} and carry out of C_{out}.

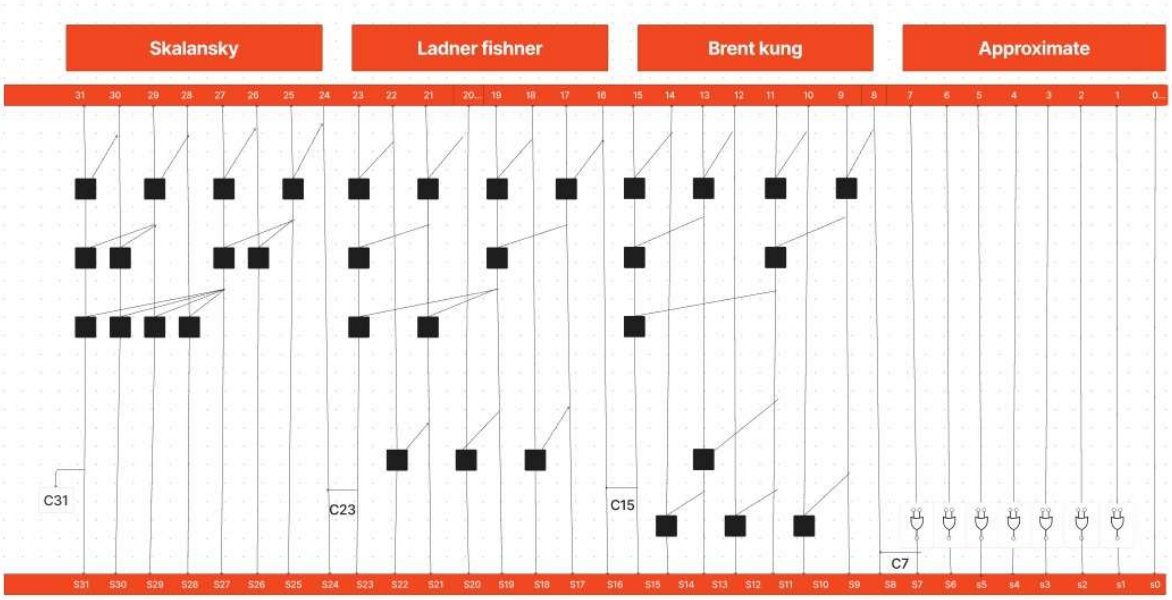

Fig. 5.10 Proposed Hybrid Approximate PPA19

5.11 PROPOSED HYBRID APPROXIMATE PPA20

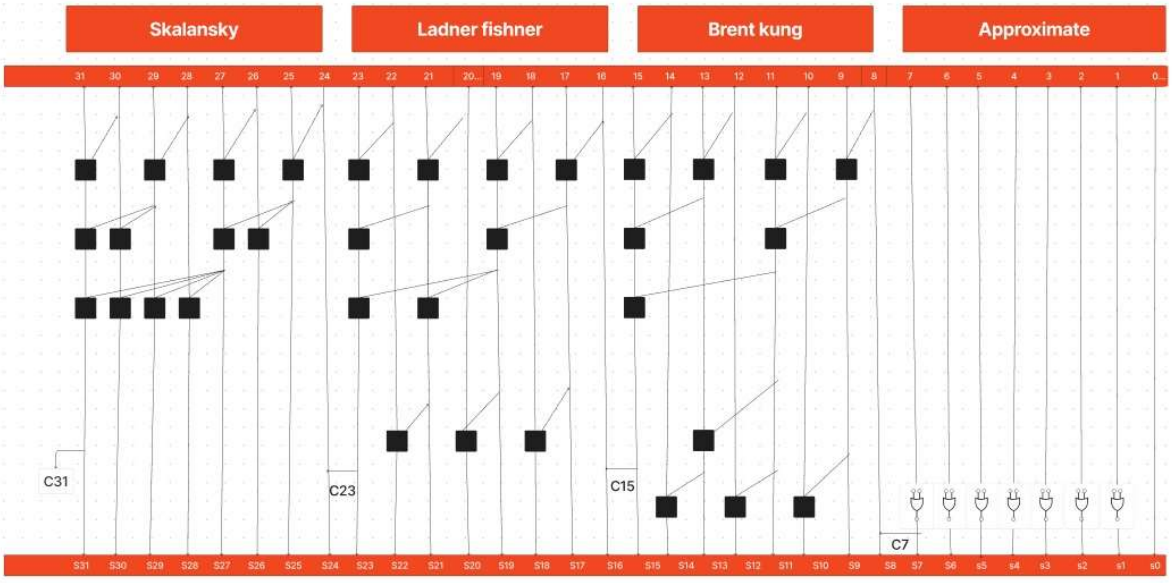

Fig. 5.11 Proposed Hybrid Approximate PPA20

The Proposed Hybrid approximate PPA20, shown in Figure 5.11, is a 32-bit adder that is divided into Four phases. It comprises of three distinct Exact PPAs and one Approximate Structure. The approximate 8-bit structure, which takes bits from LSB 0 to 7, of inputs A and B, produces

approximate sum output vales from S_0 to S_7 using the logical OR operation in the final stage and carry output C7. This carry output C_7 is used as carry input for the next stage of exact Han-Carlson 8-bit adder. The Han-Carlson 8-bit adder structure, taking bits from 8 to 15 of inputs A and B, generates a sum output with Exact values from S_8 to S_{15} and carry out of C_{15}. This carry out C_{15} is taken as input for the next exact Ladner-Fischer 8-bit Adder. The Ladner-Fischer 8-bit adder structure, taking bits from 16 to 23 of inputs A and B, generates a sum output with exact values from S_{16} to S_{23} and carry out C_{23}. The Sklansky 8-bit adder structure, taking bits from 24 to 31 of inputs A and B, generates a sum output with Exact values from S_{24} to S_{31} and carry out of $C_{out.}$

.

CHAPTER 6

IMAGE PROCESSING APPLICATIONS

6.1 INTRODUTION

Image processing refers to the manipulation and analysis of digital images using computational algorithms. It encompasses a wide range of techniques aimed at enhancing, contrast, and extracting useful information from images. The Image Processing Applications are used Image enhancement and Image constrat using FPGA In Loop.

1. IMAGE ENHANCEMENT: Image enhancement refers to a set of techniques used to improve the visual quality of digital images. The goal of image enhancement is to make images more visually appealing, easier to interpret, or better suited for specific applications. Enhancement techniques aim to adjust various image attributes such as brightness, contrast, sharpness, and colour balance.

2. IMAGE CONTRAST: Image contrast refers to the difference in brightness between the lightest and darkest parts of an image. A high-contrast image has a wide range of brightness levels, with distinct differences between bright and dark areas, while a low-contrast image has a narrower range of brightness levels, resulting in less pronounced differences between light and dark areas.

6.1.1 IMAGE ENHANCEMENT PROCESSING APPLICATION USING FPGA IN LOOP

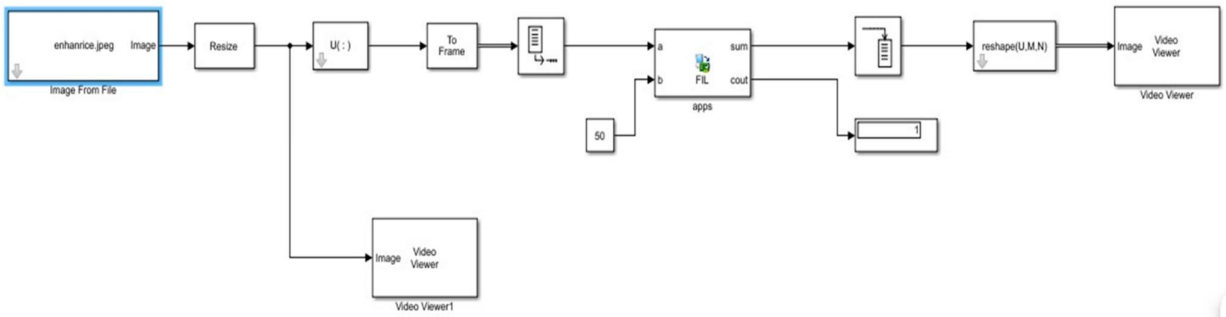

Fig. 6.1.1 Image Enhancement Processing Application Using FPGA in the Loop

The Figure 6.1.1 Image processing enhancement Application using FPGA in the Loop system, likely created using a software such as MATLAB Simulink. The flow starts with reading an image file ("enhance.jpeg"), which is then resized and processed through a series of blocks. These blocks include a transformation function U(:), converting the image to a frame, and a custom function block (F I L) that sums the input and performs other operations. The processed data is then passed through a reshape function (U,M,N) to convert it back to image format for visualization. Additionally, the original resized image is displayed using a separate Video Viewer block. The system seems to

perform an enhancement or filtering operation on the image and displays both the original and processed images for comparison. It is used for functional verification in real time system environment using FPGA in the Loop process for real time implementation on board.

6.1.1.1 Experimental Results for Image Enhancement using FPGA In Loop

Fig. 6.1.1.2 Input Image

Fig. 6.1.1.3 Output Image

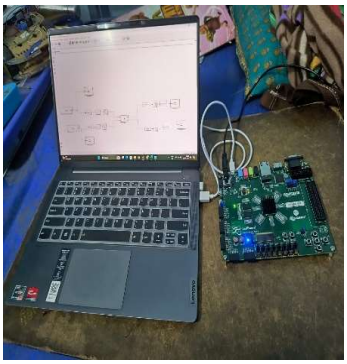

Fig. 6.1.1.4 FPGA Setup

6.1.2 IMAGE CONTRAST PROCESSING APPLICATION USING FPGA IN THE LOOP

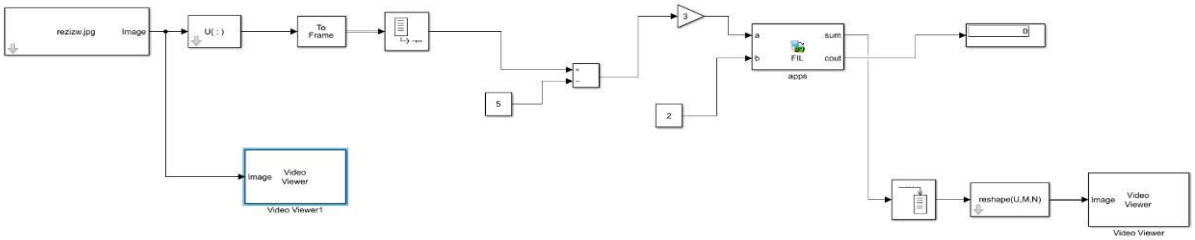

Fig. 6.1.2 Image Constrat Processing Application Using FPGA in the Loop

Image contrast is a critical parameter in image processing and computer vision that defines the difference in luminance or color that makes an object distinguishable from other objects and the background. High contrast images have a wide range of tones from black to white, making details and edges more visible and distinct. Conversely, low contrast images have a narrower range of tones, appearing more muted and less defined. Enhancing image contrast is essential for various applications, including medical imaging, remote sensing, photography, and surveillance, where clarity and detail are paramount. Techniques for contrast enhancement include histogram equalization, adaptive histogram equalization, contrast stretching, and using advanced algorithms such as CLAHE (Contrast Limited Adaptive Histogram Equalization).In a typical image processing

workflow, contrast adjustment might be one of the first steps performed after image acquisition. This can significantly improve the quality of the image, making subsequent steps like edge detection, segmentation, and object recognition more effective. The block diagrams provided likely represent a system designed to adjust and enhance image contrast, showcasing both the original and enhanced images for analysis and comparison. The Figure 6.1.2 Image Processing Constrat Application Using FPGA in the Loop illustrates an image processing workflow designed to enhance image contrast, starting with an image file labeled "resize.jpg." The image undergoes a transformation through a function U(:), then is converted to a frame for processing. The system incorporates constants (5, 3, and 2), which are combined using arithmetic operations facilitated by summation and custom function blocks (F I L). These operations adjust the image data to enhance contrast. The processed image data is then reshaped using the reshape(U,M,N) block to convert it back to an image format. The final enhanced image is displayed using a Video Viewer block, while the original resized image is also displayed separately for comparison. This workflow demonstrates a systematic approach to improving image contrast, making the details and edges more prominent, thereby facilitating better analysis and visualization

6.1.2.1 Experimental Results for Image Contrast Using FPGA in the Loop

Fig. 6.1.2.2 Input Image

Fig. 6.1.2.3 Output Image

Fig. 6.1.2.4 Input Image

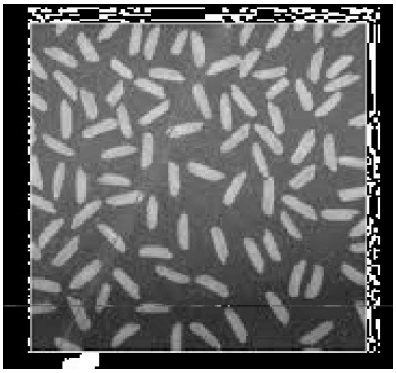

Fig. 6.1.2.5 Output Image

CHAPTER-7

SIMULATION RESULTS

7.1 INTRODUCTION

The outcomes of the Proposed Approximate PPA, Proposed Hybrid Approximate PPA, and Proposed Hybrid Approximate PPA simulations for 32-bit are displayed in Tables 7.1.1, 7.1.2, and 7.1.3, correspondingly. The implementation carried on the Artix-7 (XC7A35TICSG324-1L) FPGA board. The IEEE 1364-2005 Verilog HDL is used to model all the approximate adders. All existing and proposed adders were simulated using Xilinx Vivado 2018.1 software tool. Parameters of no. of LUTs, delay, and power consumption are measured from simulation. In the experimentation using FPGA IN LOOP, IEEE 1801-2009/2018 unified power format is used. While implementing image processing with FPGA, IEEE 1588 clock synchronization is used. To connect FPGA board to Laptop, JTAG (IEEE 1149.1-1990) is used.

7.2 SIMULATION AND SYNTHESIS RESULTS

7.2.1 Literature Review Results

The simulation results of the Exact SK, Existing AxSK, and Proposed AxSK Adders for 16 bits are shown in Figures 7.2.2,7.2.3, and 7.2.4, respectively. Based on those figures, the total values of the precise SK, AxSK, and suggested AxSK adders are 4146, 3F06, and 3F26, respectively. The two input numbers that are applied to all three adders are 8A12 and B734. The current and suggested AxSK error rates are 0,0.7053,0.6658. Table 7.2.1.4 presents a quantitative study of the area and latency in terms of LUTs for the 8-bit, 16-bit, and 32-bit AxSK adders that are already in use and those that are proposed.

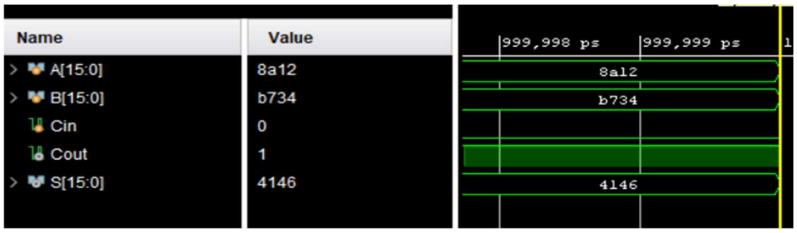

Fig. 7.2.1.1[1] Exact SK 16-Bit Adder Simulation

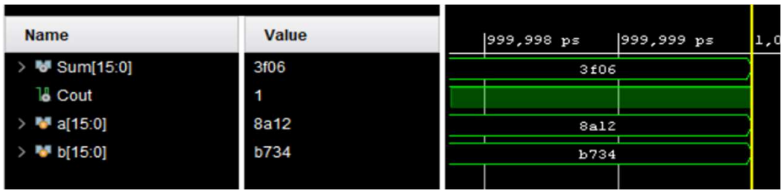

Fig. 7.2.1.2 [1] AxSK 16-Bit Adder Simulation

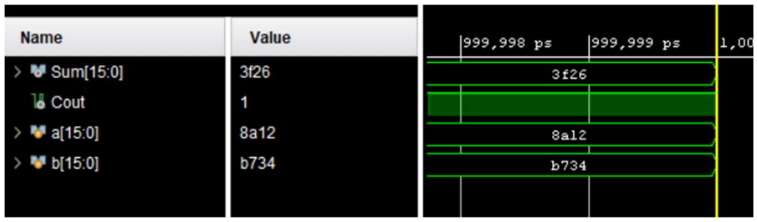

Fig. 7.2.1.3 [1] Proposed AxSK 16-Bit Adder simulation

Table 7.2.1.4 Area and Delay Analysis of the Proposed and Existing AxSK Adder

No. of bits	Exact SK Adder LUT [1]	Exact SK Adder Delay (ns) [1]	AxSK Adder LUT [1]	AxSK Adder Delay (ns)[1]	Proposed AxSKAdder LUT [1]	Proposed AxSK Adder Delay(ns) [1]
8	1386	9.234	1381	8.018	1381	8.018
16	1460	8.967	1445	7.051	1445	7.051
32	1668	9.018	1662	7.775	1662	7.775

The simulation results are shown in Figures 7.2.1.5,7.1.2.6, and 7.2.1.7 for the 32-bit versions of the Exact LF, Existing AxLF, and Proposed AxLF Adders. It is evident from those figures that 88716723 and B781ADDA are the two input numbers applied to all three adders, and that 3FF314FD, 3FF080FD, and 3FF0CAFD are the corresponding total values for the Exact LF, Existing AxLF, and Proposed AxLF Adders. The error rates of the present AxLF are 0.003147720, while the projected AxLF is 0.002794784. Table 7.2.1.8 presents a quantitative analysis of latency and area in terms of LUTs for the proposed and existing AxLF adders of 8-bit, 16-bit, and 32-bit. Table 4 illustrates that the AxLF 32-bit adder that is recommended needs less area and time.

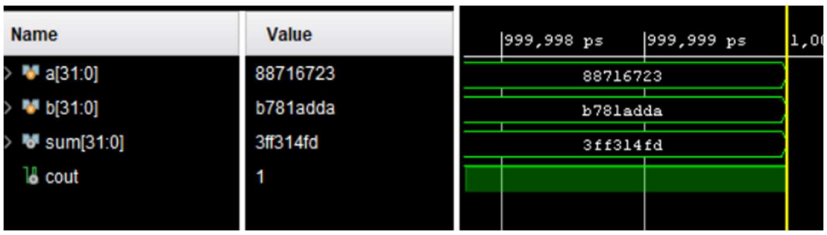

Fig. 7.2.1.5 Exact 32-bit LF Adder simulation

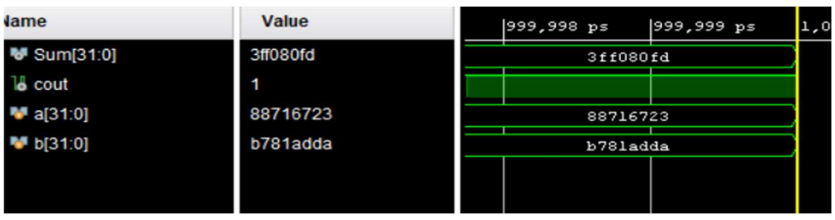

Fig. 7.2.1.6 Existing 32-bit AxLF Adder simulation

Fig. 7.2.1.7 Proposed32-bit AxLF Adder simulation

Table 7.2.1.8 Area and Time Analysis of the Existing and Proposed AxLF adders

Bits	exact LF adder LUT [1]	exact LF adder Delay (ns) [1]	AxLF Adder LUT [1]	AxLF Adder Delay (ns) [1]	Proposed AxLF adder LUT[1]	Proposed AxLF adder Delay(ns) [1]
8	1386	9.981	1383	8.837	1383	8.837
16	1454	9.463	1451	7.851	1451	7.851
32	1679	9.650	1668	8.005	1668	8.005

Results are obtained using Xilinx 14.2 tool and Cadence RTL Compiler. Path delay (ns) is calculated automatically using synthesis process in Xilinx Tool, Area and Power are Calculated using Cadence RTL Compiler Tool.

Table 7.2.1.9 Comparison of Knowles and Modified Knowles adder

Parallel prefix adders	Area [32]	Power(nw) [32]	Path delay(ns) [32]
16-bit Knowles adder	414.893	9993.106	14.452
16-bit Modified Knowles adder	340.805	8886.989	13.369

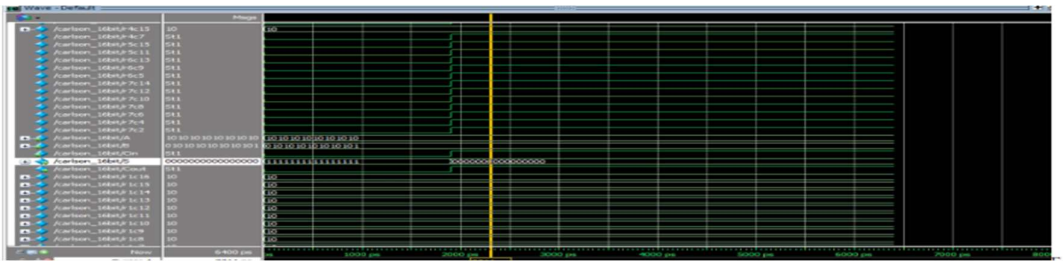

Fig. 7.2.1.10 [31] Simulation and synthesis of Han-Carlson adder

Verilog descriptions of the proposed variable latency speculative adders, and of their non-speculative counterpart. It is not easy to compare performances (in terms of power, speed, and area) of different designs, since they strongly depend on timing constraint used during synthesis. The adders we described in following discussions are CLA (Carry Look Ahead Adder) and HCA (Han Carlson Adder)

Table 7.2.1.11 Comparison of CLA Kogge stone and Han Carlson

	CLA [31]	KS [31]	HC [31]
Delay(ns)	22.95	21.82	20.82
No of Slices	1%	1%	1%
No of 4 input LUTs	1%	1%	1%
No of I/O	23%	21%	21%

Simulation results of Exact BK, AxBK, and Proposed AxBK Adders for 16 bits are shown in Figures Fig.7.2.1.12, Fig.7.2.1.13, and Fig.7.2.1.14 respectively. From those figures, we can observe that the two input numbers applied are 5AC3 and E8F9 input to all three adders, and the sum values of Exact BK, AxBK, and Proposed AxBK Adders are 43BC,43B8, and 43BA respectively. The error rates for existing and proposed AxBK are 0,0.0843,0.0434. Table 2 gives the quantitative analysis of delay and area in terms of LUTs of the existing and Proposed AxBK Adders of 8-bit, 16-bit, and 32-bit. From Table 7.2.1.15, it can be observed that the proposed AxBK 32-bit adder takes less area and less delay

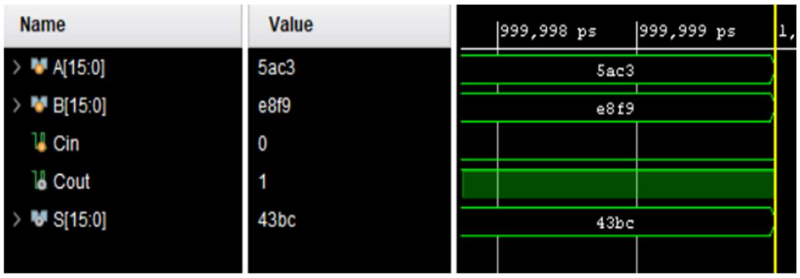

Fig. 7.2.1.12 Simulation Result of Exact BK 16-Bit Adder

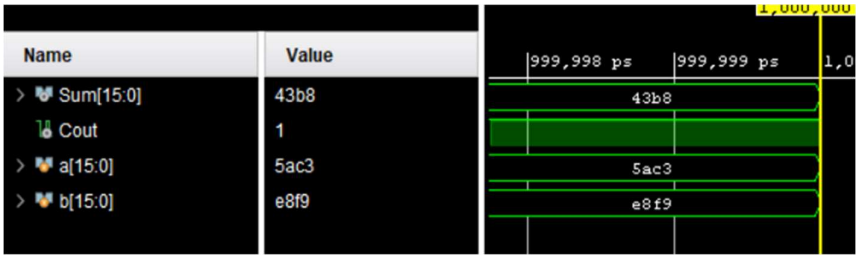

Fig. 7.2.1.13 Simulation Result of AxBK 16-Bit Adder

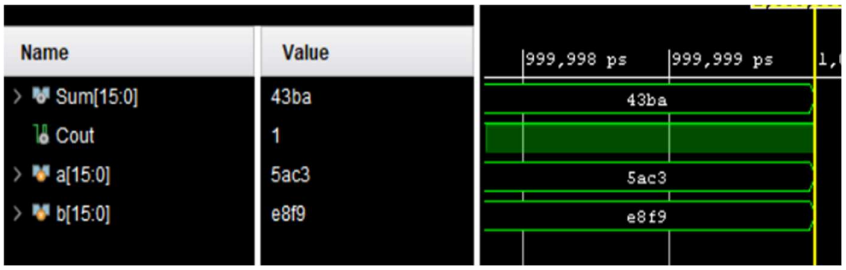

Fig. 7.2.1.14 Simulation Result of Proposed AxBK 16-Bit Adder

Table 7.2.1.15 Area and Delay Analysis of the Proposed and Existing AxBK Adder

No. of bits	Exact BK Adder LUT [1]	Exact BK Adder Delay (ns) [1]	AxBK Adder LUT [1]	AxBK Adder Delay (ns)[1]	Proposed AxBKAdder LUT [1]	Proposed AxBK Adder Delay(ns)[1]
8	1387	8.769	1388	7.861	1385	8.625
16	1460	9.583	1455	8.494	1453	8.629
32	1696	8.204	1654	7.314	1651	7.201

Simulation results of Exact KS, AxKS and Proposed AxKS Adders for 32 bits are shown in Figures Fig.7.2.1.16, Fig.7.2.1.17 and Fig.7.2.1.18 respectively. From those figures we can observe that the two input numbers applied are 8AB87B67H and B788ABDAH input to the all three adders and the sum values of Exact KS, AxKS and Proposed AxKS Adders are 42412741H, 3D308639H and 3D30D6BDH respectively. The error rates for existing and proposed AxKS are 0,1.571725781914016,1.571344539474199. Table 7.2.1.19 gives the quantitative ananlysis of delay and area in terms of LUTs of the existing and Proposed AxKS Adders of 8-bit, 16 bit and 32 bit. From the table 2, it can be observed that proposed AxKS 32-bit adder takes less area and less delay.

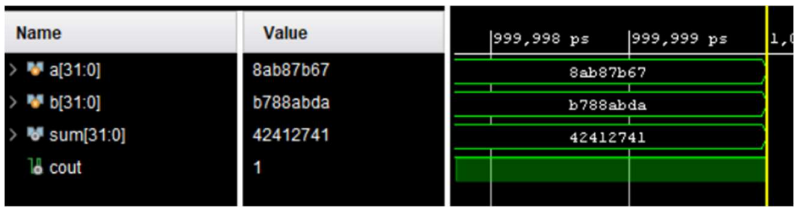

Fig. 7.2.1.16 Simulation result of 32-bit Exact KS Adder

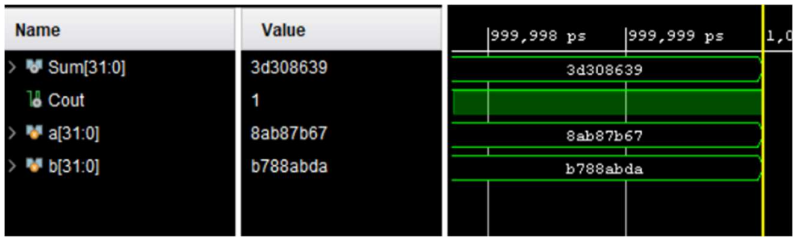

Fig. 7.2.1.17 Simulation result of 32-bit AxKS Adder

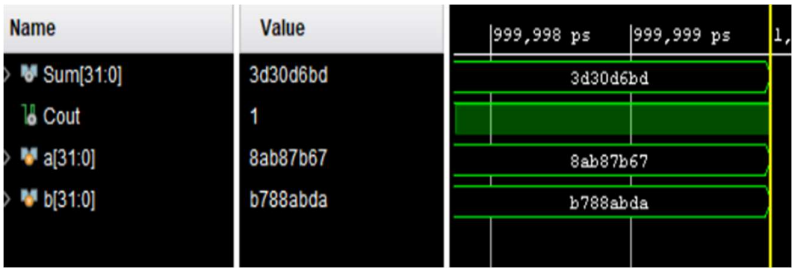

Fig. 7.2.1.18 Simulation result of 32-bit Proposed AxKS Adder

Table 7.2.1.19 Analysis of Area and Delay of existing and Proposed AxKS Adder

No. of bits	Exact KS Adder LUT [2]	Exact KS Adder [2] Delay (ns)	AxKS Adder LUT [2]	AxKS Adder Delay (ns) [2]	Proposed AxKS Adder LUT [2]	Proposed AxKS Adder Delay(ns)[2]
8	1389	8.095	1386	8.024	1386	8.022
16	1525	9.482	1453	9.067	1453	9.066
32	1659	17.460	1654	7.086	1654	7.084

7.2.2 Xilinx Vivado Simulation and Synthesis Results

Table 7.2.2.1 Performance Analysis of Proposed Approximate PPA

S. No	Types of Proposed Approximate PPA (24+8) bits	LUT	Delay (ns)
1	Proposed AxBK PPA	1663	7.532
2	Proposed AxKS PPA	1676	6.975
3	Proposed AxLF PPA	1664	7.538
4	Proposed AxSK PPA	1658	8.236
5	Proposed AxKW PPA	1679	7.983
6	Proposed AxHC PPA	1667	7.014

Table 7.2.2.1 presents the performance analysis of various types of 32-bit Proposed Approximate PPA. Among them, the Kogge Stone adder demonstrates superior speed and reduced delay compared to the others. Additionally, the Brent Kung PPA exhibits a smaller area when compared to other types of Proposed Approximate PPA.

Table 7.2.2.2 Performance Analysis of Proposed Hybrid Approximate PPA

S.NO	Types of Proposed Hybrid Approximate PPA (12+12+8) bits	LUT	Delay(ns)
1	Proposed Hybrid Approximate PPA1	1651	6.748
2	Proposed Hybrid Approximate PPA2	1658	8.021
3	Proposed Hybrid Approximate PPA3	1668	8.506
4	Proposed Hybrid Approximate PPA4	1679	8.559
5	Proposed Hybrid Approximate PPA5	1667	6.998
6	Proposed Hybrid Approximate PPA6	1662	8.260
7	Proposed Hybrid Approximate PPA7	1661	6.985
8	Proposed Hybrid Approximate PPA8	1663	7.294
9	Proposed Hybrid Approximate PPA9	1672	7.953
10	Proposed Hybrid Approximate PPA10	1662	7.195

The Performance analysis of Proposed Hybrid Approximate PPA study on various Hybrid Adders is presented in table 7.2.2.2 According to the table, the most prevalent type is Proposed Hybrid Approximate PPA1, which has a LUT size of 1651 and a delay of 6.748ns. This type exhibits lower delay and area compared to other Proposed Hybrid Approximate PPA types, while delivering high-speed performance with a lower error rate.

The table 7.2.2.3 presents the Performance analysis of Proposed Hybrid Approximate PPA study on various Hybrid Adders. Among them, the most common types are Proposed Hybrid Approximate PPA11, Proposed Hybrid Approximate PPA15, and Proposed Hybrid Approximate PPA16 with a LUT size of 1650. However, there is a slight change in delay among these three types, with the first hybrid adder having slightly lower delay compared to the other two. This particular type demonstrates lower delay and area in comparison to other Proposed Hybrid Approximate PPA types, while maintaining high-speed performance with a lower error rate.

Table 7.2.2.3 Performance Analysis of Proposed Hybrid Approximate PPA

S.NO	Types of Proposed Hybrid Approximate PPA (8+8+8+8) bits	LUT	Delay(ns)
1	Proposed Hybrid Approximate PPA11	1650	6.075
2	Proposed Hybrid Approximate PPA12	1654	7.483
3	Proposed Hybrid Approximate PPA13	1653	8.352
4	Proposed Hybrid Approximate PPA14	1653	8.152
5	Proposed Hybrid Approximate PPA15	1650	7.288
6	Proposed Hybrid Approximate PPA16	1650	7.018
7	Proposed Hybrid Approximate PPA17	1651	7.442
8	Proposed Hybrid Approximate PPA18	1659	7.467
9	Proposed Hybrid Approximate PPA19	1659	7.968
10	Proposed Hybrid Approximate PPA20	1657	7.234

7.3 IMAGE PROCESSING FOR LITERATURE REVIEW RESULTS

Output results of Exact BK, AxBK, and Proposed AxBK Adders for 16 bits are shown in Figures Fig.7.3.3,7.3.4,7.3.5,[1] respectively. From those figures, we can observe that the two input images applied are shown in Fig.7.3.1 and Fig.7.3.2 input to all three adders, and the PSNR values of Exact BK, AxBK, and Proposed AxBK Adders are calculated respectively in Table 7.3.6.

Fig. 7.3.1[1] Input Image1

Fig. 7.3.2[1] Input Image2

Fig.7.3.3[1] Exact 16-bit BKPPA

Fig.7.3.4[1] Existing 16-bit AxBK PPA

Fig.7.3.5[1] Proposed 16-bit AxBKPPA

Table 7.3.6 Calculation of PSNR (dB) for BK PPA

No of bits	Exact BK Adder [1]	AxBK Adder [1]	Proposed AxBK Adder [1]
8	46.459701	27.987721	27.908850
16	62.230587	53.218859	53.217293
32	190.974765	103.570406	103.570350

Output results of Exact KS, AxKS, and Proposed AxKS Adders for 32 bits are shown in Figures Fig.7.3.7,7.3.8,7.3.9,[2] respectively. From those figures, we can observe that the two input images applied are shown in Fig.7.3.1 and Fig.7.3.2 input to all three adders, and the PSNR values of Exact KS, AxKS, and Proposed AxKS Adders are calculated respectively in Table 7.3.10.

Table 7.3.10 Calculation of PSNR (dB) for KS PPA

No of bits	Exact KS Adder [2]	AxKS Adder [2]	Proposed AxKS Adder [2]
8	50.532650	28.817192	28.831202
16	63.333202	53.027539	53.023366
32	128.681882	101.245601	101.244375

Fig.7.3.7 Exact 32-bit KSPPA

Fig.7.3.8 Ax 32-bit KSPPA

Fig.7.3.9 Proposed 32-bit KSPPA

Output results of Exact LF, AxLF, and Proposed AxLF Adders for 32 bits are shown in Figures Fig.7.3.11,7.3.12,7.3.13,[1] respectively. From those figures, we can observe that the two input images applied are shown in Fig.7.3.1 and Fig.7.3.2 input to all three adders, and the PSNR values of Exact BK, AxBK, and Proposed AxBK Adders are calculated respectively in Table 7.3.14.

Fig.7.3.11 Exact 32-bit LFPPA

Fig.7.3.12 Ax32-bit LFPPA

Fig.7.3.13 Proposed 32-bit LFPPA

Table 7.3.14. Calculation of PSNR (dB) for LF PPA

No of Bits	Exact LF adder [1]	Existing AxLF adder [1]	Proposed AxLF adder [1]
8	50.63385	28.9066	26.888650
16	72.998084	55.712602	55.704465
32	190.9747	102.657723	102.657714

Output results of Exact SK, AxSK, and Proposed AxSK Adders for 16 bits are shown in Figures Fig.7.3.15,7.3.16,7.3.17,[1] respectively. From those figures, we can observe that the two input images applied are shown in Fig.7.3.1 and Fig.7.3.2 input to all three adders, and the PSNR values of Exact SK, AxSK, and Proposed AxSK Adders are calculated respectively in Table 7.3.18.

Fig. 7.3.15 Exact 16-bit SKPPA

Fig. 7.3.16 AxSK 16-bit PPA

Fig. 7.3.17 Proposed 16-bitSKPPA

Table 7.3.18 Calculation of PSNR (dB) for SK PPA

No of bits	Exact SK Adder [1]	Existing AxSK Adder [1]	Proposed AxSK Adder [1]
8	48.469801	26.987721	26.907850
16	111.41520	55.712602	55.704465
32	188.964765	101.561406	101.560350

7.4 IMAGE PROCESSING RESULTS USING SYSTEM GENERATOR

System Generator is a DSP design tool from Xilinx that enables the use of the MathWorks model-based Simulink design environment for FPGA design. The design tools facilitate the design processes by obscuring the technical knowledge necessary for FPGA a Register Transfer Level (RTL) design. Instead, a design is modelled using the intuitive visual environment within Simulink that uses several specific block sets accelerate the development. Additionally, System Generator can perform the FPGA implementation steps: synthesis, mapping, and place and route to generate the FPGA executable file

The image processing method need to be implemented in hardware in order to meet the real time applications. FPGA implementation can be performed using prototyping environment using Matlab/Simulink and Xilinx System Generator tool. The design flow of hardware implementation of image processing using XSG is given in Fig.7.4.1. Image source and image viewer are Simulink block sets by using these blocks image can give as input and output image can be viewed on image viewer block set. Image pre-processing and image post-processing unite are common for all the image processing applications which are designed using Simulink blocksets.

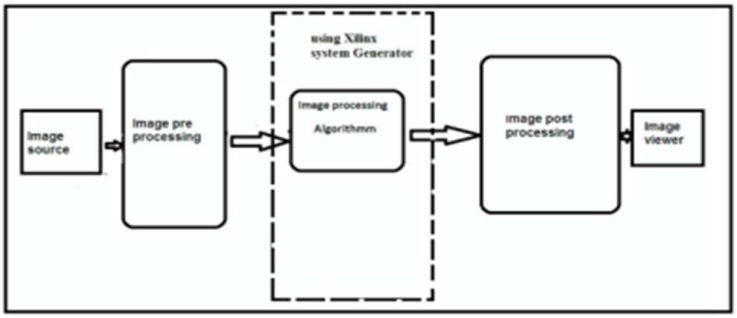

Fig. 7.4.1 Design flow of hardware implementation of image processing

A. Image Pre-Processing Unit Image preprocessing in Matlab helps in providing input to FPGA as specific test vector array which is suitable for FPGA Bitstream compilation using system generator.

- Resize: Set Input dimensions for an image and interpolation i.e. bicubic it helps in preserving fine detail in an image.

- Convert 2-D to 1-D: Converts the image into single array of pixels.

- Frame conversion and buffer: It helps in setting sampling mode and buffering of data. The model-based design used for image pre-processing is shown in Fig. 7.4.2 The blocks utilized here are discussed. Input images which could be color or grayscale are provided as input to the File block.

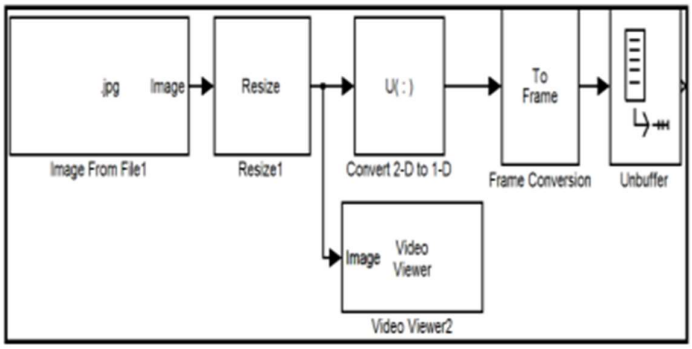

Fig. 7.4.2 Image Pre-processing unit

B. Image post processing helps recreating image from 1D array. Post-processing uses (Fig.7.4.2.1)

- Data type conversion: It converts image signal to unsigned integer format.

- Buffer : Converts scalar samples to frame output at lower sampling rate.

- Convert 1D to 2D: Convert 1D image signal to 2D image matrix.

- Video viewer: It is used to display the output image back on the monitor

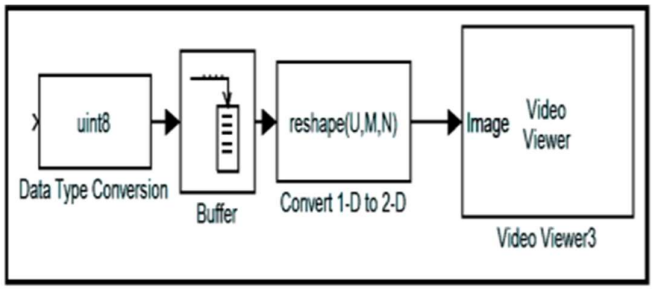

Fig. 7.4.2.1 Image Post-processing unit

Fig. 7.4.3 Input Image 1

Fig. 7.4.4 Input Image 2

Fig.7.4.5 Proposed AxBKPPA

Fig. 7.4.6 Proposed AxKSPPA

Fig. 7.4.7 Proposed AxSKPPA

Fig. 7.4.8 Proposed AxLFPPA

Fig. 7.4.9 Proposed AxKWPPA

Fig.7.4.10 Proposed AxHCPPA

Fig.7.4.11 Proposed HAxPPA1

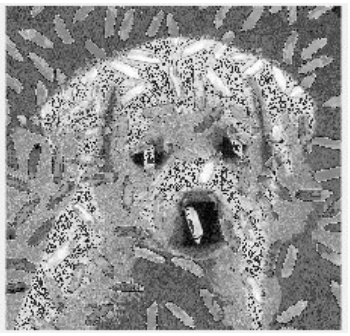

Fig.7.4.12 Proposed HAxPPA2

Fig.7.4.13 Proposed HAxPPA3

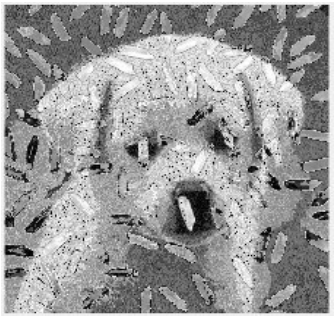

Fig.7.4.14 Proposed HAxPPA4

Fig.7.4.15 Proposed HAxPPA5

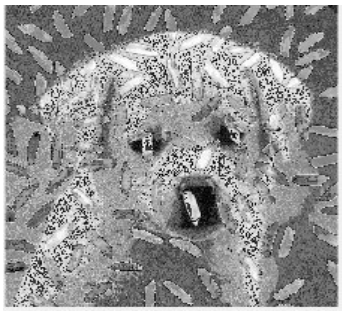

Fig.7.4.16 Proposed HAxPPA6

Fig.7.4.17 Proposed HAxPPA7

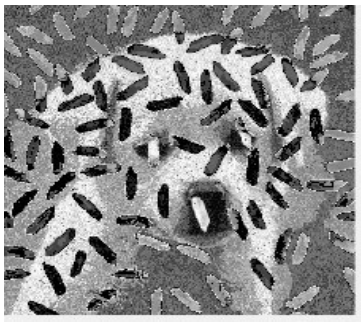

Fig.7.4.18 Proposed HAxPPA8

Fig.7.4.19 Proposed HAxPPA9

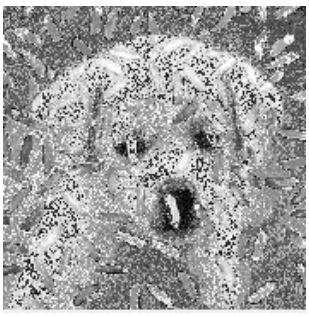

Fig.7.4.20 Proposed HAxPPA10

Fig.7.4.21 Proposed HAxPPA11

Fig.7.4.22 Proposed HAxPPA12

Fig.7.4.23 Proposed HAxPPA13

Fig.7.4.24 Proposed HAxPPA14

Fig.7.4.25 Proposed HAxPPA15

Fig.7.4.26 Proposed HAxPPA16

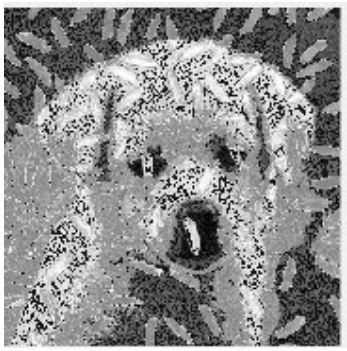

Fig.7.4.27 Proposed HAxPPA17

Fig.7.4.28 Proposed HAxPPA18

Fig.7.4.29 Proposed HAxPPA19

Fig.7.4.30 Proposed HAxPPA20

7.4.31 Calculation of PSNR Values for Proposed Approximate PPA using System Generator

Peak Signal-to-Noise Ratio (PSNR) is a crucial metric used to evaluate the quality of reconstruction in image and video compression. It quantifies the difference between an original image and a compressed or processed version, providing a measure of the fidelity and accuracy of the compression algorithm. Typically expressed in decibels (dB), a higher PSNR value indicates a closer resemblance to the original image, implying better quality. PSNR is widely used due to its simplicity and the intuitive understanding it offers: higher PSNR equates to lower error and, therefore, higher visual

quality. However, it should be noted that PSNR may not always correlate perfectly with perceived visual quality, as it is primarily a mathematical measure rather than a perceptual one.

Table 7.4.31 Calculation of Proposed Approximate PPA PSNR Values

S. No	Types of Proposed Approximate PPA (24+8) bits	PSNR (dB)
1	Proposed Approximate BK PPA	107.060868
2	Proposed Approximate KS PPA	107.836712
3	Proposed Approximate LF PPA	107.060868
4	Proposed Approximate SK PPA	107.182541
5	Proposed Approximate KW PPA	107.263412
6	Proposed Approximate HC PPA	107.468125

Table 7.5.1 is Based on the PSNR values for different types of Proposed Approximate Parallel Prefix Adders (PPA), it is evident that all the evaluated PPAs exhibit excellent performance in image reconstruction, with PSNR values exceeding 107 dB. Among them, the Proposed Approximate KS PPA stands out with the highest PSNR of 107.836712 dB, indicating superior quality and minimal error. The Proposed Approximate HC PPA follows closely, also demonstrating high reconstruction quality. Proposed Approximate KW and SK PPAs show comparable performance with PSNR values of 107.263412 dB and 107.182541 dB, respectively, while Proposed Approximate BK and LF PPAs, both with a PSNR of 107.060868 dB, slightly lag behind. Despite these differences, all PPAs maintain a high level of accuracy, reflecting their effectiveness in preserving image fidelity during compression.

7.4.32 Calculation of PSNR for Proposed Hybrid Approximate PPA

Table 7.4.32 Shows the PSNR values for various types of Proposed hybrid approximate PPAs reveal significant insights into their performance in image reconstruction. Among the evaluated hybrids, "Proposed Hybrid Approximate PPA4" stands out with the highest PSNR of 108.430561 dB, closely followed by "Proposed Hybrid Approximate PPA10" at 108.391740 dB, indicating superior image quality and minimal error. "Proposed Hybrid Approximate PPA2" and "Proposed Hybrid Approximate PPA8" also demonstrate high performance with PSNR values of 107.960343 dB and 107.803920 dB, respectively. The remaining PPAs, although slightly lower in PSNR, still maintain excellent reconstruction quality, with values all above 106.94 dB. Overall, the Proposed hybrid

designs effectively enhance image fidelity, with certain combinations like Proposed HAxPPA4 and Proposed HAxPPA10 providing particularly high accuracy.

Table 7.4.32 Calculation of PSNR for Proposed Hybrid Approximate PPA

S.NO	Types of Proposed Hybrid Approximate PPA (12+12+8) bits	PSNR (dB)
1	Proposed Hybrid Approximate PPA1	107.456310
2	Proposed Hybrid Approximate PPA2	107.960343
3	Proposed Hybrid Approximate PPA3	107.458650
4	Proposed Hybrid Approximate PPA4	108.430561
5	Proposed Hybrid Approximate PPA5	107.242392
6	Proposed Hybrid Approximate PPA6	106.948555
7	Proposed Hybrid Approximate PPA7	107.216459
8	Proposed Hybrid Approximate PPA8	107.803920
9	Proposed Hybrid Approximate PPA9	107.215912
10	Proposed Hybrid Approximate PPA10	108.391740

7.4.33 Calculation of PSNR for Proposed Hybrid Approximate PPA using System Generator

The PSNR values for various types of Proposed Hybrid Approximate PPAs, based on the given data, provide insight into their effectiveness in image reconstruction. "Proposed Hybrid Approximate PPA12" achieves the highest PSNR of 109.417188 dB, indicating the best image quality and lowest error among the tested combinations. This is followed by "Proposed Hybrid Approximate PPA17" and "Proposed Hybrid Approximate PPA20" with PSNR values of 108.455380 dB and 108.412348 dB, respectively, also demonstrating excellent performance. The majority of other hybrids, including "Proposed Hybrid Approximate PPA11", "Proposed Hybrid Approximate PPA14", and "Proposed Hybrid Approximate PPA16", show strong performance with PSNR values around 107 dB, ensuring high-quality image reconstruction. However, "Proposed Hybrid Approximate PPA13" presents a notably lower PSNR of 102.972126 dB, indicating poorer image quality.

Table 7.4.33 Calculation of PSNR for Proposed Hybrid Approximate PPA

S.NO	Types of Proposed Hybrid Approximate PPA (8+8+8+8) bits	PSNR (dB)
1	Proposed Hybrid Approximate PPA11	107.089177
2	Proposed Hybrid Approximate PPA12	109.417188
3	Proposed Hybrid Approximate PPA13	102.972126
4	Proposed Hybrid Approximate PPA14	107.089177
5	Proposed Hybrid Approximate PPA15	106.392073
6	Proposed Hybrid Approximate PPA16	107.067981
7	Proposed Hybrid Approximate PPA17	108.455380
8	Proposed Hybrid Approximate PPA18	107.004355
9	Proposed Hybrid Approximate PPA19	107.117702
10	Proposed Hybrid Approximate PPA20	108.412348

CHAPTER-8

VERIFICATION RESULTS

8.1 INTRODUCTION

Field Programmable Gate Arrays (FPGAs) are powerful hardware devices widely used for implementing real-time, high-performance image processing tasks. Their ability to handle parallel processing and high data throughput makes them ideal for image processing applications where speed and efficiency are critical. Here, we will introduce the concept of using FPGAs in a loop for image processing, detailing the workflow and verification process shown in Fig. 8.1 Image Processing Using FPGA In Loop. The implementation carried on the Artix-7 (XC7A35TICSG324-1L) FPGA board. In the experimentation using FPGA IN LOOP, IEEE 1801-2009/2018 unified power format is used. While implementing image processing with FPGA, IEEE 1588 clock synchronization is used. To connect FPGA board to Laptop, JTAG (IEEE 1149.1-1990) is used.

8.2 IMAGE PROCESSING USING FPGA IN LOOP

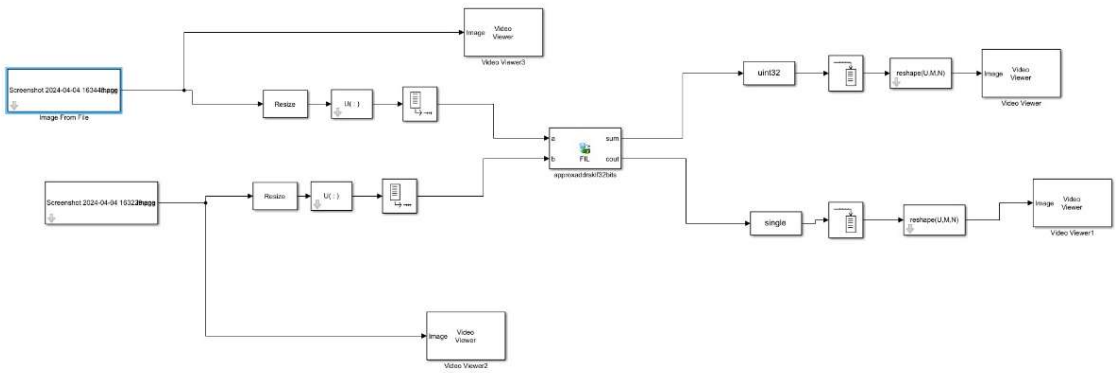

Fig. 8.2 Image Processing Using FPGA In Loop

The image you sent depicts a block diagram of a system for processing images. but here's a general breakdown of the steps involved:

1. **Image Input**: The system takes an image as input, which is denoted by a block labeled "[text in the image not shown] Image." This image can be in various formats, such as JPEG, PNG, or BMP.
2. **Data Type Conversion**: The image data is then passed through a block labeled "Data Type Conversion." This block likely converts the image data from its original format into a format that the system can process. For instance, the image data might be converted from a high-level format like JPEG into a simpler binary format.

3. **Image Resizing**: Next, the image data goes through a block labeled "Image Resize." This block may resize the image to a specific resolution. Resizing an image involves changing the number of pixels in the image. This can be done for various reasons, such as to reduce the file size of the image or to match the image to the size requirements of the system.
4. **Single/reshape (MN)**: The block labelled "Single/reshape (MN)" likely performs two operations on the image data. The "Single" operation might convert the data type of the image from double-precision floating-point numbers to single-precision floating-point numbers. Single-precision numbers use less memory than double-precision numbers, but they also have a lower precision (meaning they can represent a smaller range of values). The "reshape (MN)" operation likely reshapes the image data into a two-dimensional matrix with M rows and N columns. This is a common way to store and process image data.

1. **Video Viewer**: The output from the previous block is then fed into a block labeled "Video Viewer." It's important to note that the system is processing an image, not a video. The "Video Viewer" block might be a generic term for a block that can display the processed data. In this case, it would be displaying the image data.
2. **Data Type Conversion**: The system also has another block labeled "Data Type Conversion." The purpose of this block is likely similar to the first "Data Type Conversion" block. It might convert the processed image data back into its original format or into a different format for further processing.
3. **Output**: The final block in the system is labeled "Out." This block represents the output of the system, which could be the processed image data.

Overall, the block diagram depicts a system that can process images in various ways, including resizing the image, converting the data type, and potentially performing other operations on the image data.

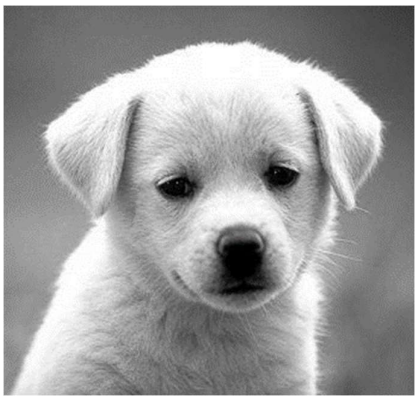

Fig. 8.2.1 Input Image 1

Fig. 8.2.2 Input Image 2

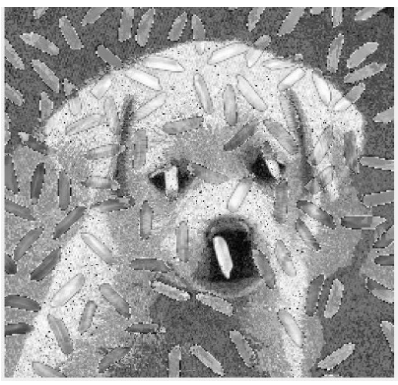

Fig. 8.2.3 Output Image

CHAPTER 9

CONCLUSION AND FUTURE SCOPE

9.1 CONCLUSION

Various approximation adders have been examined, simulated, and analyzed in this study. The experimental findings were then implemented. Within this project, novel approximate parallel prefix adders were devised and implemented to attain decreased area and delay, along with high PSNR values. This addition method is exceptionally swift and can be executed for large numbers in significantly less time, yielding superior outcomes. The proposed Ax parallel Prefix adders exhibit potential applications in digital signal processing, ALU units, and Image Processing. In this project, image enhancement and image contrast applications are implemented.

9.2 FUTURE SCOPE

The focus is on further research on hybrid PPAs, architecture modifications, and specific image processing applications and implementation.

9.3 REFERENCES

[1] R. P. Brent and H. T. Kung, “A regular layout for parallel adders,” Compute. Eng., vol. C-31, IEEE, Trans., vol. C-31, no. 3, pp. 260-264, 1982 [doi:10.1109/TC.1982.1675982].

[2] Dr.M. Sarada Musala, Satish Kanapala, Sudhakar Reddy Dantla, Prudhvi Tummala, “Approximate Brent Kung Adder for Image Processing Applications.”, Proceedings of 2023, Springer.

[3] Dr.M. Sarada Musala, Satish Kanapala, Sudhakar Reddy Dantla, Prudhvi Tummala, “High Speed Area Efficient Approximate Kogge Stone Adder.”, IEEE Proceedings of 2023.

[4] Y. Choi, “Parallel prefix adder design” in Proc. 17th IEEE Symposium on Computer Arithmetic, Jun. 27 2005, pp. 90-98.

[5] T. Lynch and E. E. Swartzlander, "A spanning tree carry look-ahead adder," IEEE Trans. Compute., vol. 41, no. 8, pp. 931-939, Aug. 1992 [doi:10.1109/12.156535].

[6] B. C. Geetha and D. M. Lohitha, “Navya and Pramod,” Perform. Anal. Parallel Prefix Adder Datapath Des., IEEE Xplore Compliant - Part Number: CFP18BAC-ART; ISBN: 978-1-5386-1974-2, p. V.

[7] M. M. Azevedo da Rosa et al., “AxPPA: approximate parallel prefix adders,” IEEE Trans. Very Large Scale Integrated. (VLSI) Syst., vol. 31, no. 1, January 2023.

[8] P. da Costa et al., " Improved approximate multipliers for single-precision floating-point hardware design,' in Proc. IEEE" 13th Latin Am. Symp. Circuits Syst. (LASCAS), Mar. 2022, pp. 1-4.

[9] “Comparative analysis of parallel prefix adders,” Megha Talsania and Eugene John.

[10]"Parallel prefix adder design," IEEE Trans. Compute., 2001, Andrew Beaumont- Smith and Cheng–Chew Lim.

[11] A. Kumar et al., “Design and study of adder multiplier by using 4:2 compressors and parallel prefix adder for VLSI-circuit design” 2nd International Conference for Emerging Technology (INCET) Belgaum, India May, vols. 21-23, 2021.

[12] M. Prasanna Kumar et al., "'Comparative Analysis of Brent Kung and Kogge Stone Parallel Prefix Adders for their Area, Delay and Power Consumption', research paper engineering," ISSN, vol. 5, no. 10, Oct., p. 2249555, 2015.

[13] P. P. Potdukhe and V. D. Jaiswal, Design of High-Speed Carry Select Adder Using Brent Kung Adder [doi:10.1109/ICEEOT.2016.7754762].

[14] N. Udaya Kumar and K. Bala, Sindhuri; K. Durga Teja; D. Sai Satish, Implementation and comparison of VLSI architectures of 16-bit carry select adder using Brent Kung adder.

[15] K. Golda, Hepzibah; CP. SubhaA Novel Implementation of High-Speed Modified Brent Kung Carry Select Adder.

[16] G. G., S. S. Raju, and S. Suresh, "Parallel Prefix Speculative Han Carlson Adder," in IOSR Journal of Electronics and Communication.

[17] B. Koyada et al., "A comparative study on adders" in IEEE Conference on Wireless Communication, Signal Processing and Networking, 2017 [doi:10.1109/WiSPNET.2017.8300155].

[18] B. Mohamad et al., "Template matching using the sum of squared difference and normalized cross-correlation" in Proc. IEEE Student Conference Res. Develop. (Scorned), Dec. 2015, pp. 100-104.

[19] M. M. A. da Rosa et al., "'Exploring efficient adder compressors for the power-efficient sum of squared differences design,' in Proc.," Circuits Syst. (ICECS) 27th IEEE Int. Conf. Electron., Nov. 2020, pp. 1-4.

[20] A. V. Gupta et al., Low-Power Digital Signal Processing Using Approximate.

[21] S. K. Yezerla and B. R. Naik, “Design and Estimation of delay power and area for Parallel prefix adders.”, Proceedings of, 2014, RAECS, 06-08 March, 2014

[22] N. H. E. Weste et al., VLSI Design. Pearson: Addison-Wesley, 2011.

[23] S. Sri Katyayani, Dr.M. Chandramohan Reddy, Murali.k, “Design of Efficient Han-Carlson-Adder”, (IJET).

[24] Pawan Kumar1, Jasbir Kaur2” Design of Modified Parallel Prefix Knowles Adder” International Journal of Science and Research (IJSR) ISSN (Online): 2319-7064 Impact Factor (2012): 3.358.