JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

DESIGN OF HIGH SPEED THREE OPERAND ADDER USING PARALLEL PREFIX ADDERS

[1] Chamarthi Tejaswini, [2] K.Babitha,M.Tech
[1]M.Tech PG Scholar, [2]Asst.Professor
[1,2]Deptartment Of ECE,
[1,2] SKR College of Engineering and Technology,Konduru satram (v),Manubolu(m),SPSR Nellore(dt),Andhra Pradesh.

Abstract: In this paper the aim is to perform high speed three operand addition. Three operand binary adders are the basic functional unit to perform modular arithmetic in various cryptography and pseudorandom bit generator(PRBG) algorithms.Binary adders are known as important elements in the circuit designs. Many fastest adders have been created and developed. Parallel Prefix Adders (PPA) are one among them. We use adders frequently in digital design and VLSI designs, in digital design we use adders such as half adder, full adder which are serial type. By using both adders we can perform addition for any number of bits. But it has a huge delay problem. To overcome this, Parallel Prefix Adders are preferred. In VLSI implementation parallel prefix adders are known to have the best performance. This paper presents an implementation of various types of carry tree adders like the Kogge- Stone, Brent Kung, Han Carlson, and Ladner Fischer and compares them with serial adders. Hence, a new high speed adder architecture is proposed.

I.INTRODUCTION

Binary addition is one of the most fundamental calculations performed by today's microprocessors (Patterson & Hennessy, 1994). Binary addition is performed in much the same way as decimal addition. Each digit of the vector is added, from right to left, producing a sum and a carry. The sum of two binary digits is very simple to calculate, and can be expressed in a simple truth table, often called a "half-adder" (Wakerley, 2000):

Input 1	*Input 2*	*Carry*	*Sum*
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Table 1.1: Binary Half-Adder Truth Table

In order to calculate the addition of a vector longer than one bit, this table must be expanded to include the addition of the carry output of the previous bit. This is expressed in the following truth table, often called a "full-adder" (Wakerley, 2000):

Carry-In	*Input 1*	*Input 2*	*Carry-Out*	*Sum*
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Table 1.2: Binary Full-Adder Truth Table

To provide a method for handling negative numbers, binary vectors are typically expressed in a form called the 2s complement. The 2s complement uses the most significant bit to indicate the sign (0 is positive, 1 is negative), and requires a specific bit length to be chosen the 2s complement is calculated by performing a 1s complement of the vector that describes the positive number, and adding 1. An example of this would be to model -7 in a 4-bit number would result in the following conversion. As a 4-bit binary number, '7' is expressed as '0111'. First, we perform a Is complement to give' 1000'. Finally, we add one to give' 1001'. By quick examination, we can determine that a 2s complement number of bit length N will result in a number in the range: _2N - 1 <= number < 2N1.

In most addition schemes, there is no direct way to perform subtraction to bypass this limitation, the complement of the number to be subtracted is added. The 2s complement is a quick and easy method to implement to perform subtraction. For example, instead of subtracting '55' from '77', '-55' is added to '77' (Patterson & Hennessy, 1994)

Because of the limitations of the specific bit sizes in microprocessors, specific tests must be added to watch for overflow conditions, or conditions that result in an incorrect result. These conditions occur whether the addition is signed or not This is because any addition of numbers of bit sized N can potentially result in an answer that needs (N+ 1) bits to describe. For example, if you add the unsigned 3-bit numbers of '111' and '001' together, you end up with '1000', a 4-bit number.

Microprocessor complexity has grown quickly, and the width of the binary vectors used has increased from 8-bits to 32-bits, 64-bits, and beyond. Although the sum may be calculated by a binary adder quickly, the inability of most binary adders to determine whether the addition is complete requires the worst case to be assumed. The worst-case addition is based upon the length of time for the carry calculation to propagate across the entire width of the sum vector. As such, addition of vectors of width N can require N operations to calculate despite the fact that the average case has but 10~N carry propagation. Therefore, tremendous research has gone into reducing the time required to perform that addition The research presented here investigates an innovative implementation of a binary adder, hereafter called a carry-feedback adder. The carry-feedback adder is compared against five traditional binary adder designs. The following five designs were chosen because they are either fundamental or highly optimized: carry-ripple, carry-completion, carry-look ahead, carry-select, and pyramid adder schemes. The design comparison metrics are speed and size that are combined to determine overall performance.

There is tremendous difficulty in establishing fairness in regard to these two metrics, as the exact implementation of a design often depends on the transistor technology being used. In an effort to maintain simplicity, factors such as 'fan-out' (a single output attached to too many gate inputs) and 'fan-in' (too many inputs to logic gate) are ignored. As such, speed is measured in terms of gate delays, and the gates are assumed to all require the same amount of time, represented as t. Likewise, size is measured in gate count, and gates are all assumed to be the same size. Adder variables are chosen such that the values regarding fan-in and fan-out are reasonable.

The number of gate delays is calculated by determining the worst-case path through the adder. For example, given the equation AB + CD, the number of gate delays is two: one to determine the outputs of the gate (A AND B) and (C AND D), which are calculated simultaneously, another to determine the output of the OR gate that combines them. Likewise, the gate count is determined by summing up the total number of gates in the adder (XOR is counted as a single gate). For example, given the equation (AB + CD), the number of gates is three: one for AB, one for BC, and one for AB + CD. Adder performance is measured by multiplying speed and size. Adders with lowers values for this calculated parameter indicate a higher performance. This research weights speed and size equally in regards to performance, but it is a simple change to weight one metric more important than another

For all adder designs and comparisons, the vectors A and B denote the addends, the vector C denotes the carries, and the vector S denotes the sum. All of the vectors are of width

N. A subscript of a vector, such as An, represents the single bit of vector A at position n, where n ranges from 0 to (N - 1). Cin is a single bit input to the adder representing the carry input, sometimes represented as Co. Cout is a single bit output of the adder representing the carry output, sometimes represented as CN. Where latching is required, two technologies are compared. The first is a traditional D flip-flop; the second is an SRAM cell. D flip-flops have 6 gates and require 3t gate delays to latch the output (Wakerley, 2000). SRAM cells have 1 gate equivalent and require 1 t gate delays to latch the output. Both of these technologies are used to provide an adequate comparison for implementations where SRAM cells, the obviously better choice, are not available.

II. PROJECT DESCRIPTION

Addition is one of the fundamental arithmetic operations and it has been used extensively in many VLSI systems such as microprocessors, DSP and other specific application architectures. In addition to its main task, which is adding two numbers, it is the nucleus of many other useful operations such as, subtraction, multiplication, address calculation and etc. It is also the speed limiting and more power consuming element as well. The design of faster, smaller and more efficient adder architecture has been aim and goal for many research efforts and has resulted in a large number of adder architectures. Each architecture provides different insight and thus suggests different implementations. The power consumption and propagation delay are two most important properties of the adder circuit architectures which basically are against each other. That knows, lowering the power causes longer propagation delay and vice versa, hence, most architectures referring to one of those important properties. Nevertheless, in some cases they booth may compromised to achieve to low energy consumption. All architectures provide different insight and therefore require different implementation. This chapter provides overall and essential information and abstract of the most adder architectures in system level. In general, full Adder function can introduce either using Boolean logic function (conventional architecture) or the majority-function.

Boolean Logic Full Adder Function

A full adder Boolean logic function is based on three inputs, (Ai, Bi, Ci-1) and provides two outputs Si and Ci.

$$S_i = A_i \oplus B_i \oplus C_{i-1}$$

$$C_i = A_i B_i + B_i C_{i-1} + A_i C_{i-1}$$

However, it is most common and practical to use denoted characters Pi (Carry Propagate) and Gi (Carry Generate).

$$P_i = A_i + B_i$$

$$G_i = A_i B_i$$

So, sum and carry outputs is given by:

$$C_i = G_i + P_i C_{i-1}$$

$$S_i = P_i \oplus C_{i-1}$$

There are different solutions to implement n-bits full adder. First, architecture must be defined based on speed or power consumption. Then logic cell implementing will take place to finish the design cycle. In following section, we will have quick look at the most common n-bit full adder architectures.

Boolean Logic Full Adder Architectures

Since today many types and architectures are introduced for the full adder Boolean logic but all they can conceptually categorized into two major groups:

(a) Carry Propagate (CP)
(b) Carry Look-Ahead (CLA)

In first group (group "a"), generated carry propagates from first to the last digit or it may skip some blocks. Based on equitation (3.2.6), result of sum (Si) depends on this propagation. These groups consider having a linear base structure which requires less logic gates and so less power consuming. But it has long delay in compare with group "b". Glitch in this group also is a consequence of such a delay or propagation time in big adder and it has an impact on dynamic power consumption. Most full adder architectures design fall in to this group ("a"). Some of the very popular architectures in this group are:

1. Ripple Carry Adder.
2. Carry Skip Adder.
3. Carry Select Adder.
4. Carry Save Adder

The second group (group "b") however calculates the carry in advance to avoid that propagation time delay. For every bit, the (Si) is independent of last sum result so the ripple effect has thus been terminated. Because of that termination, the number of bits won't change addition time and it will be independent than bits numbers. However, due to increasing number of overhead gates in these circuits, propagation delay increases but still this group is faster than the first one in operation under certain condition. As a result of increasing number of gates, this group also considers high power consuming again in certain condition that we will see later. Some of the most popular architectures in this group are:

1. Carry look ahead adder
2. Kogge-Stone Adder.

3.3.1 Ripple Carry Adder

The simplest addition architecture is based on a linear array of a full adder cell as it is depicted in figure (3.3.1). This architecture which also known as RCA has been subjected to be the smallest and the lowest power consuming. However according to the experimental results in this model, they show the average activity overhead (glitch) is about 50% [2]. The worst-case delay or the critical delay path in N-Bit RCA is given by: where tcarry is the carry propagation delay from the input to the output

$$t_p = (N-1)t_{carry}$$

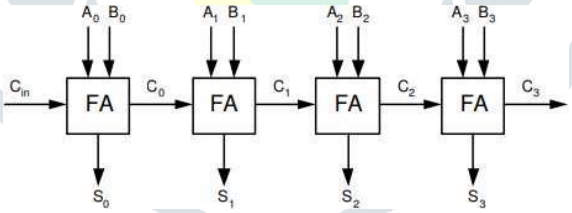

Figure 3.1: 4-bit Ripple Carry Adder.

Carry Skip Adder

The carry skip adder like RCA is based on linear structure but it takes advantage of extra gates to skip from designated blocks that logic gates are leading to make long critical path Cin to Cout shorter. Hence the critical propagation delay path in N-bit full adder when it is divided to M-bits groups is given by,

$$t_p = t_{pg} + Mt_{carry} + (\frac{N}{M} - 1)t_{skip}$$

Where tpg is the required time to generate "P" and "G" signals and tcarry is the carry propagation delay in each group and tskip is the multiplexer propagation delay. Note that multiplexer propagation delay in equation (3.3.2) consists of select signal generating delay (N/M bits AND logic) and data path delay.

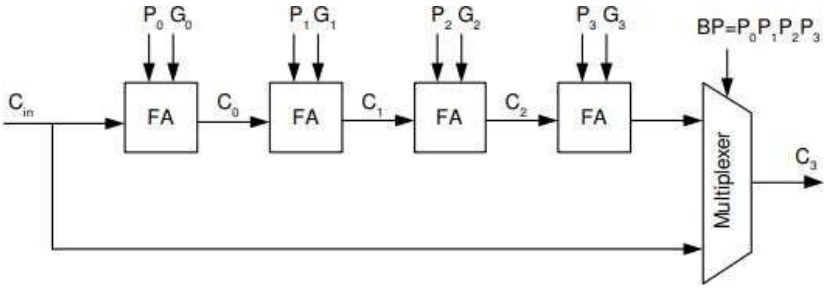

Figure 3.2: 4-bit Carry Skip Adder.

Carry Select Adder

The main idea in a carry select adder is to split a sequential adder into two parts and performing the computation of most significant bit (MSB) part with considering the two possibilities for carry-in bit in parallel. The right generated carry then will be selected using the carry-out bit of the least significant bit (LSB). In this case the critical delay path in N-bit full adder when it is divided to M-bit group is given by,

$$t_p = Mt_{carry} + (\frac{N}{M})t_{multiplexer}$$

$$M = \sqrt{\frac{N}{2}}$$

Where tcarry is the carry propagation delay in each group and tmultiplexer is the multiplexer propagation delay. Figure (3.3.3) shows the idea of carry select adder.

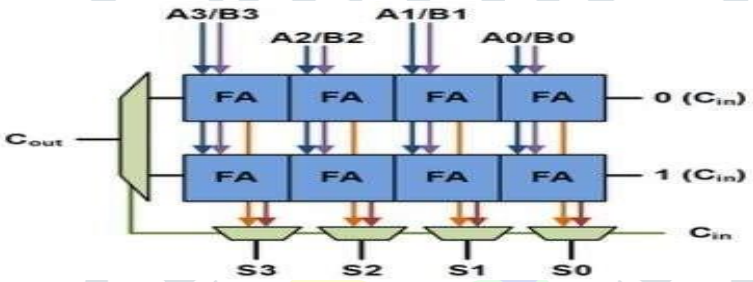

Figure 3.3: 4-bit Carry Select Adder.

Carry Save Adder

There are many cases where it is desired to add more than two numbers together. The straightforward way of adding together N numbers (all M bits wide) is to add the first two, then add that sum to the next, and so on. This requires a total of (N − 1) additions, for a total gate delay of (N.log M). Using carry save adder, the delay can be reduced further still. The idea is to take these three numbers that we want to add together, X + Y+ Z, and convert it into two numbers C + S such that X + Y + Z = C + S. The carry save adder consists of a ladder of standalone full adders, and carries out a number of partial additions. The principal idea is that carry has a higher power of two and thus is routed to the next column. Doing additions with Carry save adder saves time and logic. Figure 3.4 shows the general idea of carry save adder with four operands.

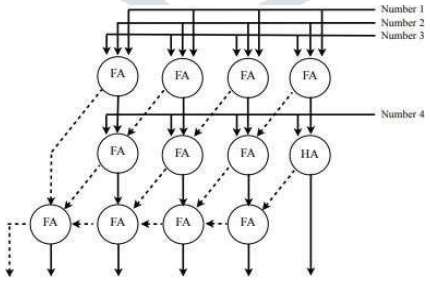

Figure 3.4: 4-operands Carry Save Adder.

Carry save adder which also known as CSA comparison with the other standard adder is not straightforward. CSA systems require more bits than the other for the same interval of higher delay (two gates delay) than the HanCarlson adder, with a 10% to 18% reduction in the gate complexity [22].

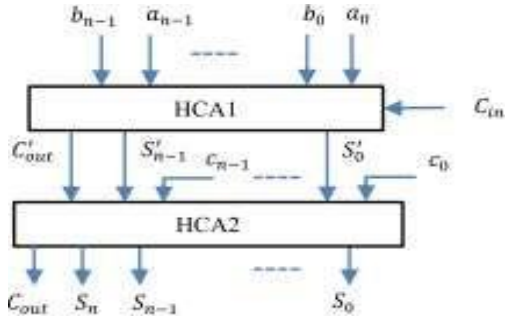

Figure 3.8: Block level architecture of HCA-based three-operand adder (HC3A).

PROPOSED THREE-OPERAND ADDER ARCHITECTURE

This section presents a new adder technique and its VLSI architecture to perform the three- operand addition in modular arithmetic. The proposed adder technique is a parallel prefix adder. However, it has four-stage structures instead three-stage structures in prefix adder to compute the addition of three binary input operands such as bit-addition logic, base logic, PG (propagate and generate) logic and sum logic.

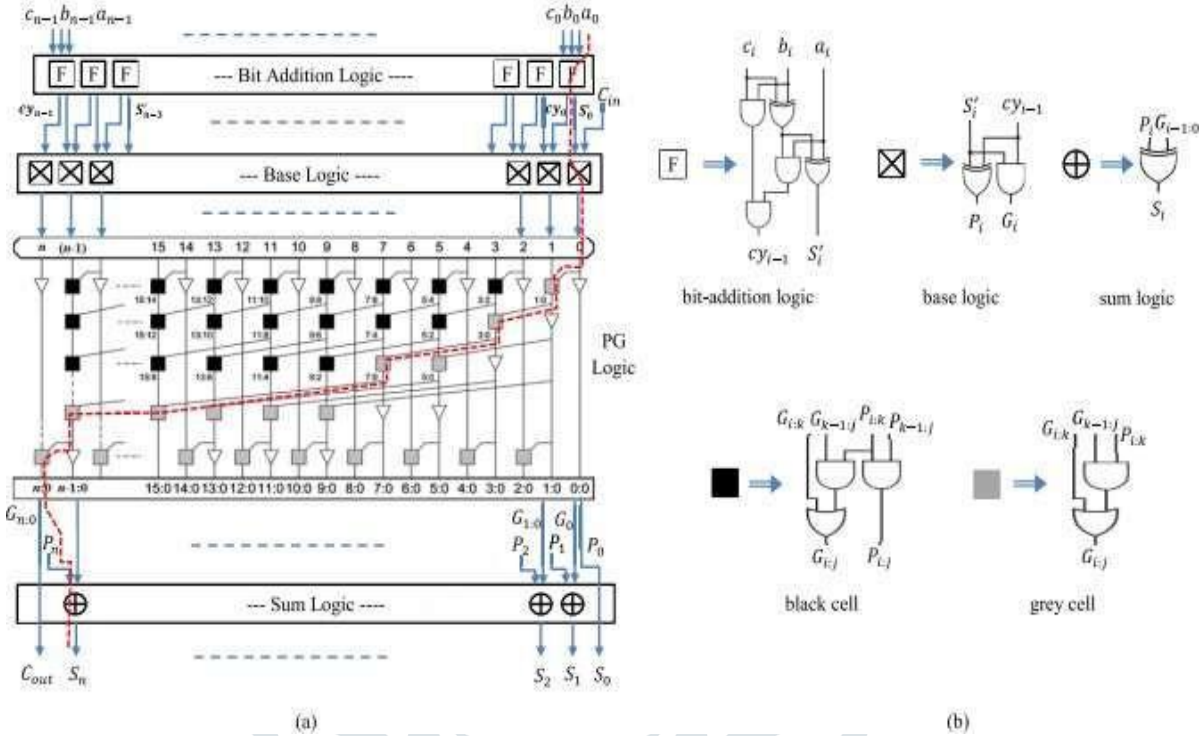

Figure 3.9: Proposed three-operand adder;a.)First order VLSI architecture, b.)Logical diagram of bit addition, base logic, sum logic, black-cell and grey-cell.

The proposed VLSI architecture of the three-operand binary adder and its internal structure is shown in Fig. 3. The new adder technique performs the addition of three n-bit binary inputs in four different stages. In the first stage (bit-addition logic), the bitwise addition of three n- bit binary input operands is performed with the array of full adders, and each full adder computes “sum (i)” and “carry (cyi)” signals as highlighted in Fig. 3.9(a). The logical expressions for computing sum (i) and carry (cyi) signals are defined in Stage-1, and the logical diagram of the bit-addition logic is shown in Fig. 3.9(b).

In the first stage, the output signal “sum (i)” bit of current full adder and the output signal “carry” bit of its right-adjacent full adder are used together to compute the generate (Gi) and propagate (Pi) signals in the second stage (base logic). The computation of Gi and Pi signals are represented by the “squared saltire-cell” as shown in Fig. 3.9(a) and there are n +1 number of saltire-cells in the base logic stage. The external carry-input signal (Cin) is also taken into consideration for three-operand addition in the proposed adder technique. This additional carry-input signal (Cin) is taken as input to base logic while computing the G0 (S0 · Cin) in the first saltire-cell of the base logic. The third stage is the carry computation stage called “generate and propagate logic” (PG) to pre-compute the carry bit and is the combination of black and grey cell logics. The logical diagram of black and grey cell is shown in Fig. 3.9(b) that computes the carry generate Gi: j and propagate Pi: j signals

The number of prefix computation stages for the proposed adder is (log2 n+1), and therefore, the critical path delay of the proposed adder is mainly influenced by this carry propagate chain.

IV.Results

Existing system:

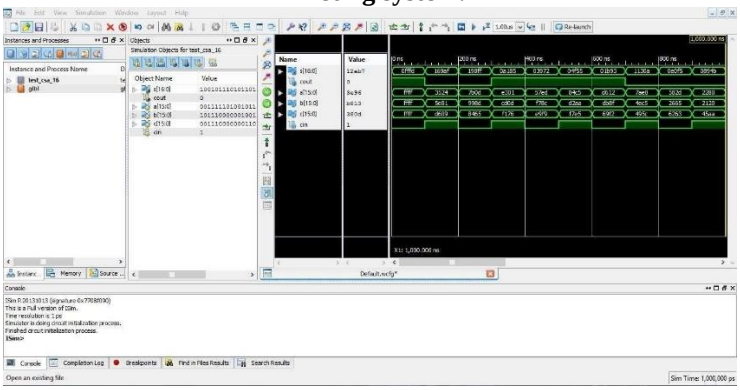

Figure 4.1: Simulation result of the existing system

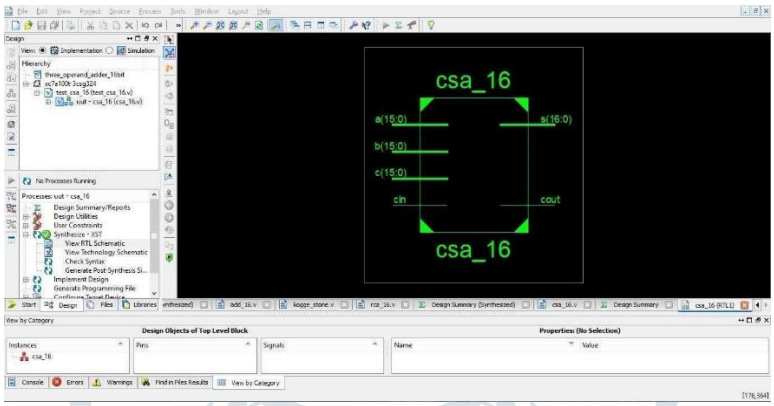

Figure 4.2: Block diagram of the existing system

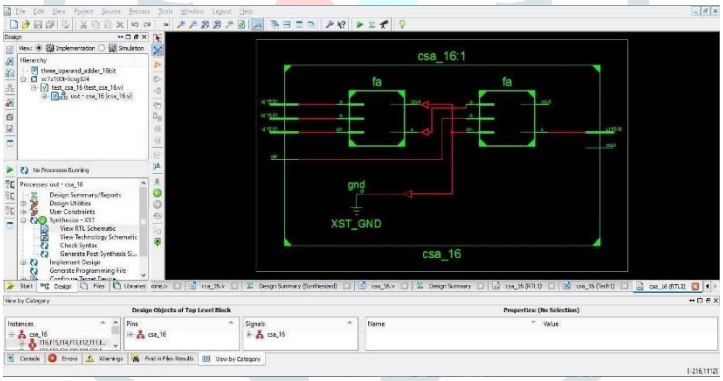

Figure 4.3: RTL schematic of the existing system

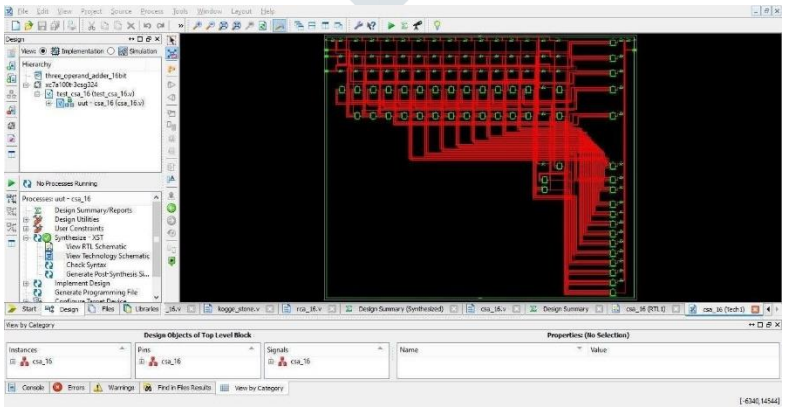

Figure 4.4: Technology schematic of the existing system

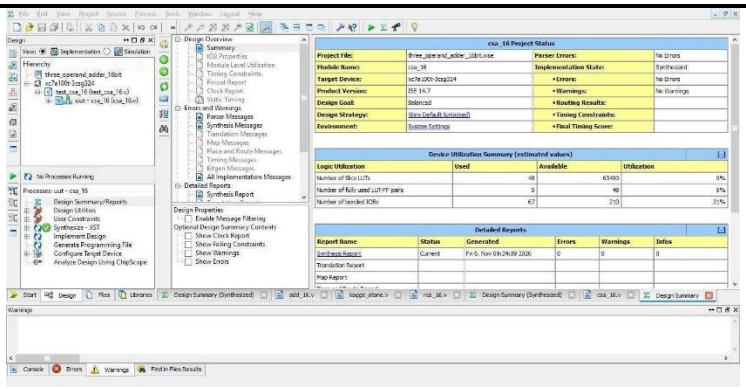

Figure 4.5: Summary report of the existing system

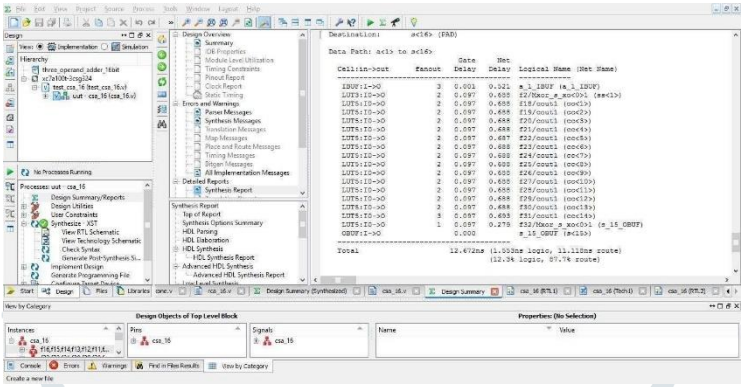

Figure 4.6: Delay report of the existing system

Proposed system:

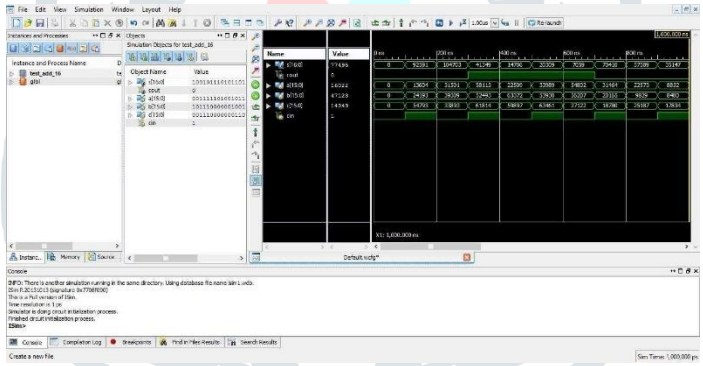

Figure 4.7: Simulation result of the proposed system

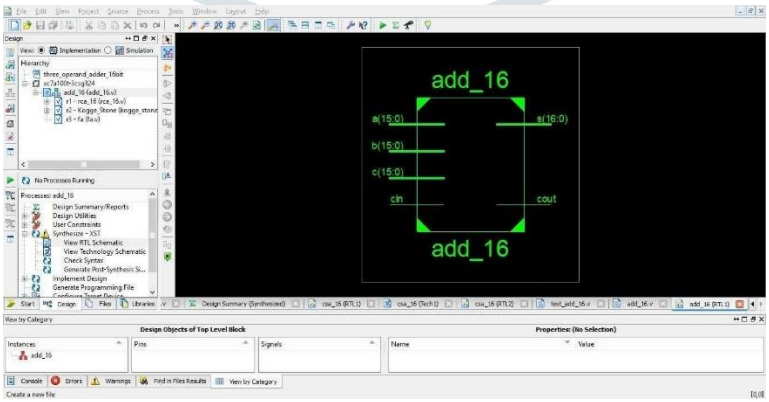

Figure 4.8: Block diagram of the proposed system

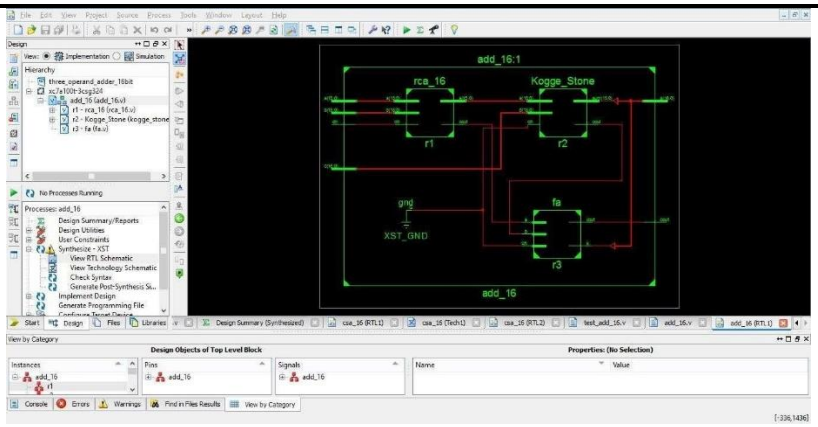

Figure 4.9: RTL schematic of the proposed system

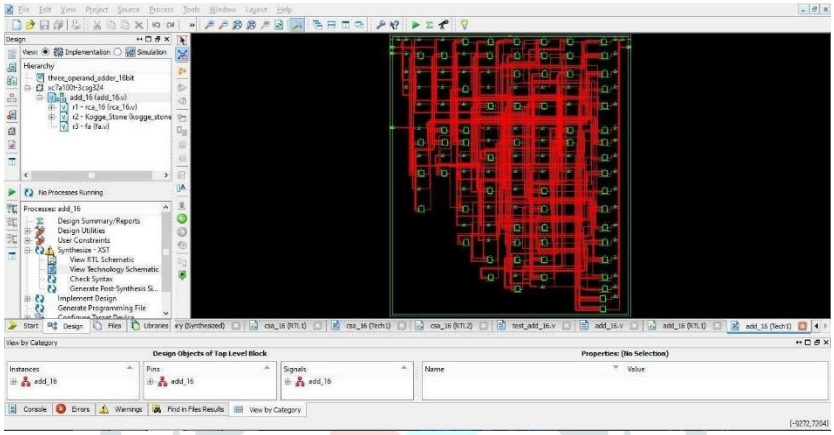

Figure 4.10: Technology schematic of the proposed system

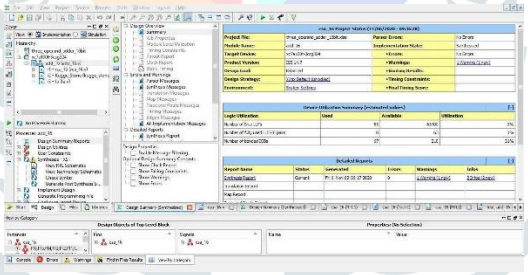

Figure 4.11: Summary report of the proposed system

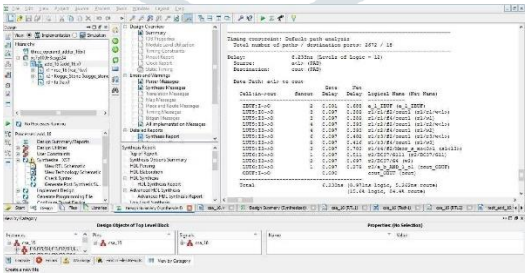

Figure 4.12: Delay report of the proposed system

Comparison results:

Method	Adder Name	Delay in ns	Area in LUT
Existing	**Carry save adder**	**12.672 ns**	**48**
Proposed	**Kogge- stone adder**	**6.233 ns**	**63**

Table 4.1: Comparison results of existing & proposed methods

V.Conclusion & Future scope:

In this paper, a high-speed adder technique and its VLSI architecture is proposed to perform the three operand binary addition for efficient computation of modular arithmetic used in cryptography and PRBG applications. The proposed three-operand adder technique is a parallel prefix adder that uses four-stage structures to compute the addition of three input operands. The novelty of this proposed architecture is the reduction of delay in the prefix computation stages in PG logic and bit-addition logic that leads to an overall reduction in critical path delay and power-delay product (PDP).

For the fair comparison, the concept of hybrid Han-Carlson two-operand adder is extended to develop a hybrid Han-Carlson three-operand adder (HHC3A) topology. The same coding style adopted in proposed adder architecture is extended to implement the HHC3A, HC3A and CS3A using Verilog HDL.

Further, all these designs are synthesized using commercially available 32nm CMOS technology library to obtain the core area, timing and power for different word size. From the physical synthesis results, this is clear that the proposed adder architecture is 3 to 9 times faster than the corresponding three operand carry save adder (CS3A) adder architecture.

Moreover, a sharp reduction in timing path and power dissipation can be observed in the proposed adder as compared to the carry save adder (CS3A). But area increases in proposed adder architecture, this may be reduced by using other parallel prefix adders.

VI.REFERENCES

1. Jasbir Kaur, Lalit Sood “Comparison between various Types of Adder topologies” International journal Computer Science and Technology Vol.6, March 2015.
2. Jasmine Saini, Somya Agarwal, Aditi Kansa “Performance, Analysis and Comparison of Digital Adders “International Conference on advances in Computer Engineering and Application (ICACEA) 2015.
3. R.P.P.Singh, Parveen Kumar “Performance Analysis Fast Adders using VHDL” International conference on Advances in recent technologies in communication and Computing2009.
4. Nidhi Tiwari, Ruchi Sharma, Rajesh Pariha “Implementation Of Area and energy efficient full adder Cell” IEEE International Conference on recent advances and innovation in Engineering 2014.
5. Rajkumar Sarma and Veerati Raju “Design and Performance Analysis of hybrid adders for High speed arithmetic circuit” International Journal of VLSI Design and communication Systems (VLSICS) Vol 3, no 3 June2012.
6. Arkadiy Morgenshtein, Alexander fish and Israel Wagner “Gate Diffusion input (GDI)
   - a technique for low power Design of Digital circuits Analysis and characterization IEEE International symposium on circuit and system February2012.
7. Sentamilselvi M, Mahendran P “High performance Adder Circuit in VLSI system” International journal of Technology Enhancements and emerging engineering Research IJTEEE VOL 2 Issue3.
8. Shambhavi Mishra, Gaurav Verma “Low power and Area efficient implementation of BCD Adder of FPGA” IEEE 2013.
9. Rashmi Chawla, Prashant kumar, Priya Yadav “Adde Circuit Design using advanced quantum dot cellular Automata (AQCA)” National Conference on Recent Advances in electronics and Computer engineering RAECE 2015.