

Design and Implementation with Area, Power and Delay Estimation of Hybrid Parallel Prefix Adder

Tanveer Fatima¹, Sree Uma²

¹Department of ECE, Siddhartha Institute of Technology and Sciences, Hyderabad, India (PG Scholar)

²Department of ECE, Siddhartha Institute of Technology and Sciences, Hyderabad, India (Associate Professor)

ABSTRACT

Conventional adders like BINARY ADDITION, HALFADDER (HA), FULLADDER (FA), Ripple Carry Adder (RCA), Carry Look ahead Adder (CLA) and Carry Skip Adder (CSA) cannot satisfy optimization goals so, in this paper we proposed four types of Parnell prefix adders like Kogge Stone Adder (KSA), Spanning Tree Adder (STA), Brent Kung Adder (BKA) and Sparse Kogge Stone Adder (SKA) to achieve high speed and area optimization. Then we are generated a new hybrid adder using KSA and BKA. this adder takes less area and low power. These adders are designed using (VHSIC) Hardware Description Language (HDL) and simulation, synthesis using Xilinx Integrated Software Environment (ISE) 13.2 Design Suite. These designs are implemented in Xilinx SPARTAN 3E Field Programmable Gate Arrays (FPGA).

Keywords: Kogge Stone Adder, Brent Kung Adder, Sparse Kogge Stone Adder, Spanning Tree Adder, hybrid adder.

INTRODUCTION

Adders are commonly found in the critical path of many building blocks of microprocessors and digital signal processing chips. Adders are essential not only for addition, but also for subtraction, multiplication, and division. Addition is one of the fundamental arithmetic operations. A fast and accurate operation of a digital system is greatly influenced by the performance of the resident adders. The most important for measuring the quality of adder designs in the past were propagation delay, and area. Adders are the basic building blocks of all arithmetic circuits; adders add two binary numbers and give out sum and carry as output. Basically we have two types of adders. The design of a binary adder begins by considering the process of addition in base 2, In this example, the two numbers to be added, 1101 + 0110, are written one above the other. The carries from one position to the next (called internal carries) are written above them. The result is the 5-bit number 10011. Interpreting the values of the binary numbers, this sum corresponds to the decimal addition $13 + 6 = 19$. From this example, we observe that the problem of adding two 4-bit numbers can be reduced to the problem of adding a column of two or three bits and passing the carry along to the next column. (For the sake of regularity, we can all in a 0 as the input carry in the rightmost column, so then each column is always the sum of three bits.) A circuit that adds a column of three bits is called a full-adder. (The name full-adder comes from the fact that it can be constructed by combining two half-adders, each of which adds only two bits, in a way that we shall see shortly.) We can design such a Circuit by making a table listing the outputs for all possible input combinations. Note that in each column there is a sum bit, which is put at the bottom, and a carry bit that is taken to the next column. So we need to specify two output bits for each input combination. Let us call the two data bits x and y .

A full-adder can be constructed from two half-adders and an OR gate. The explanation of why this works is as follows. (In this paragraph, $+$ denotes addition, not the OR operation.) Consider the addition of $x + y + z$. This can be grouped as $(x + y) + z$ where $(x + y)$ represents the output of the half-adder that receives x and y . This partial sum is added to z by the other half-adder, yielding the

**Address for correspondence:*

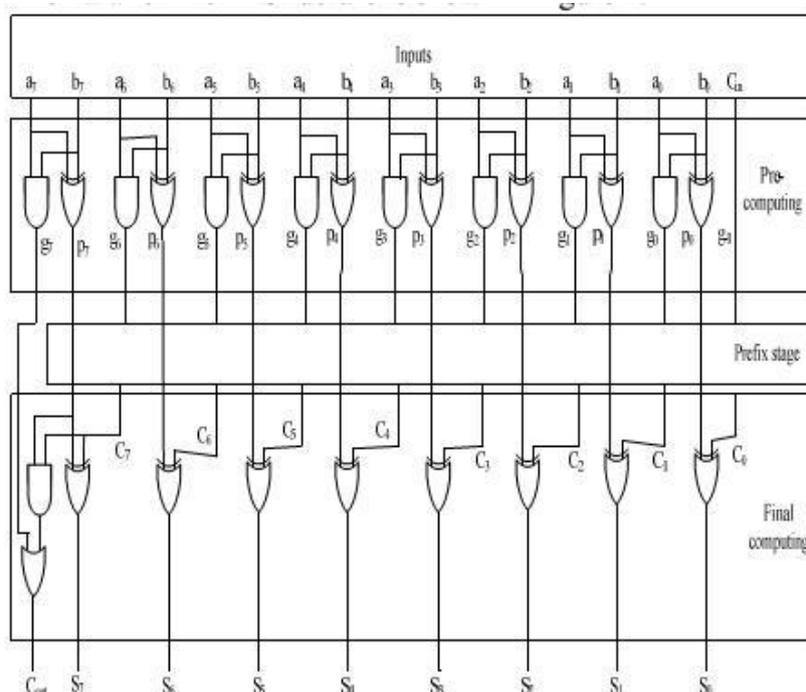
tanu.hy21@gmail.com

complete sum bits. As for C, consider that there are two possible ways to make $C = 1$: rst, if $x + y = 2$, then adding z can only make the total sum 2 or 3, and either way $C = 1$. In this case, the rst half-adder's carryout is a 1. Second, if $x + y = 1$, then C will be 1 only if $z = 1$ to make the total sum 2. In this case, the second half-adder's carry output will be 1. Thus we see that $C = 1$ if and only if at least one of the half-adders produces a carryout of 1. This corresponds to the OR of the two partial carry bits. Now, to complete the design we need only construct the half-adder out of basic gates. The straightforward design methodology will not yield the simplest design. Instead we use some cleverness that will allow c and s to share some logic. With the full-adder design in hand, we can now construct an n -bit adder simply by stringing full-adders together. Each full-adder adds the bits in one bit position, say i , where $i = 0; 1; n1$. Thus the i -th full-adder receives the data bits A_i and B_i from the two numbers to be added. It also receives a carry-in C_i from the full-adder in the next-lower-numbered bit position. It produces bit S_i of the sum, and sends a carryout C_{i+1} to the full-adder in the next-higher-numbered bit position. The full-adder for the least-significant bit, $i = 0$, is an exception: it receives its carry-in from an external source. The full-adder for the most-significant bit, $i = n - 1$, is also an exception: its Carry-out is sent out externally as the end-carry C_n Thus, improving the speed of addition will improve the speed of all other arithmetic operations. Accordingly, reducing the carry propagation delay of adders is of great importance. Different logic design approaches have been employed to overcome the carry propagation problem. One widely used approach employs the principle of carry look-ahead solves this problem. By calculating the carry signals in advance, based on the input signals. This type of adder circuit is called as carry look-ahead adder (CLA adder). It is based on the fact that a carry signal will be generated in two cases: when both bits A_i and B_i are 1, or when one of the two bits is 1 and the carry-in (carry of the previous stage) is 1. To understand the carry propagation problem, let's consider the case of adding two n -bit numbers A and B . Generates all the P & G signals. Four sets of P & G logic (each consists of an XOR gate and an AND gate). Output signals of this level (P 's & G 's) The Carry Look-Ahead (CLA) logic block which consists of four 2-level Implementation logic circuits. It generates the carry signals (C_1, C_2, C_3 , and C_4) as defined by the above expressions. Output signals of this level (C_1, C_2, C_3 , and C_4) Four XOR gates which generate the sum signals (S_i) ($S_i = P_i \oplus C_i$). Output signals of this level (S_0, S_1, S_2 , and S_3).

PARALLEL-PREFIX ADDER STRUCTURE

PPA's basically consists of 3 stages

- Pre computation
- Prefix stage
- Final computation

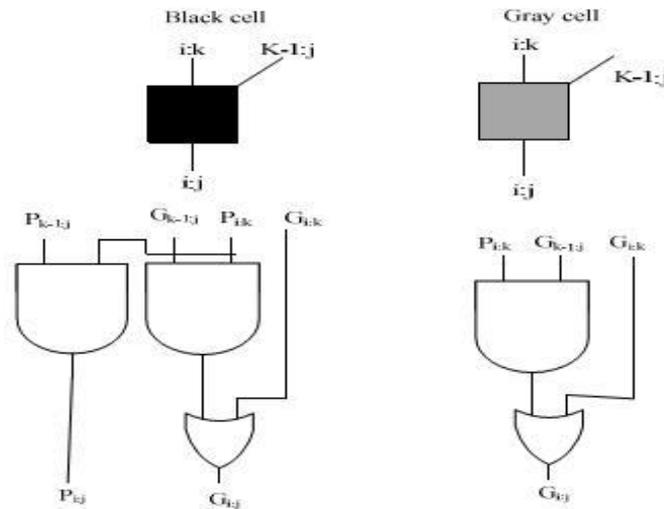


In pre computation stage, propagates and generates are Computed In the prefix stage, the black cell (BC) and gray cell (GC) generates only the building prefix structures. Final computation, the sum and carryout are the final output.

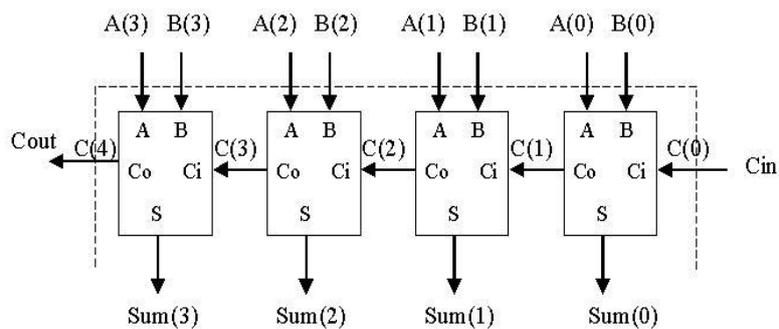
Black Cell and Gray Cell

In black cell having four inputs and two outputs propagation means and operation and generation means and-or operation.

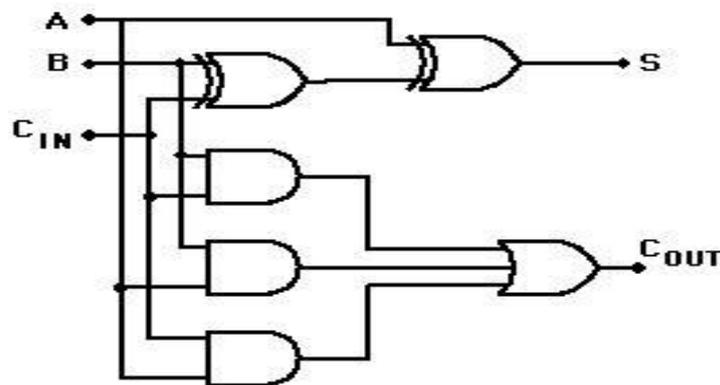
In gray cell having three inputs and one output here generation means and-or operation.



Ripple Carry Adder

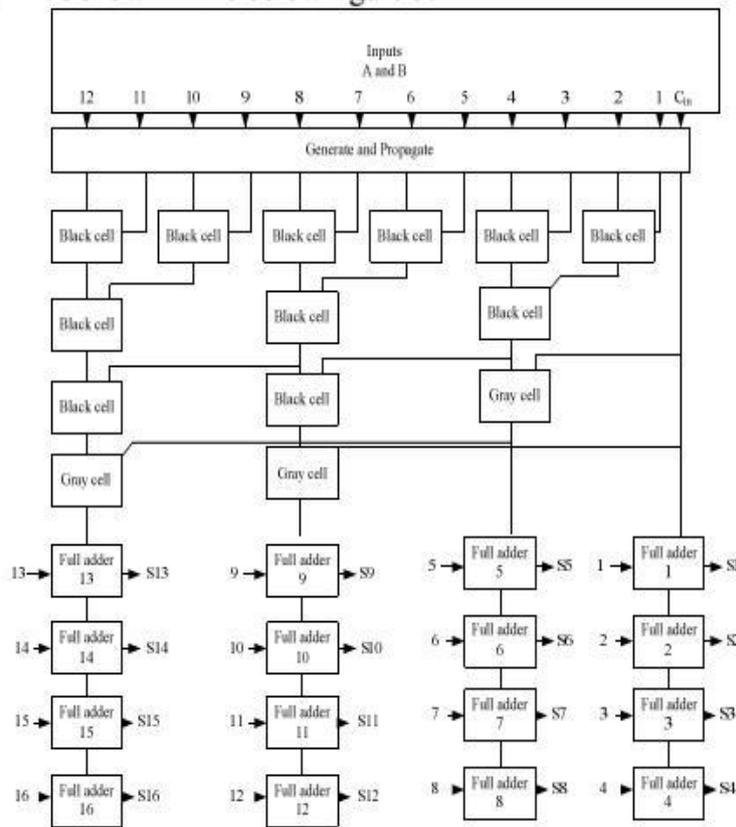


Full adder



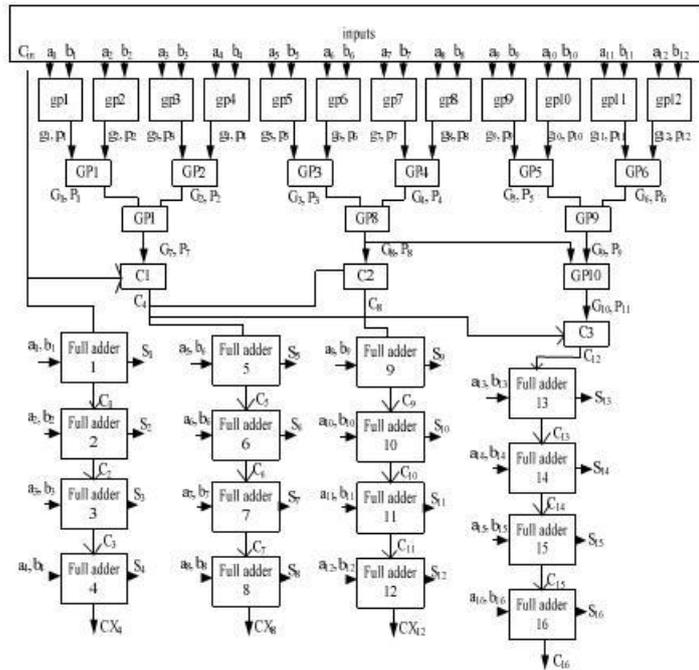
16 Bit Sparse Kogge-Stone Adder

In this adder initially pre computation stage we are doing propagation and generation operations. After that stage1 used 6 black cells. Stage2 used 3 black cells. Stage3 used. 2 black cell and one gray cell.stage4 used two gray cells. Then apply ripple carry adder operation.



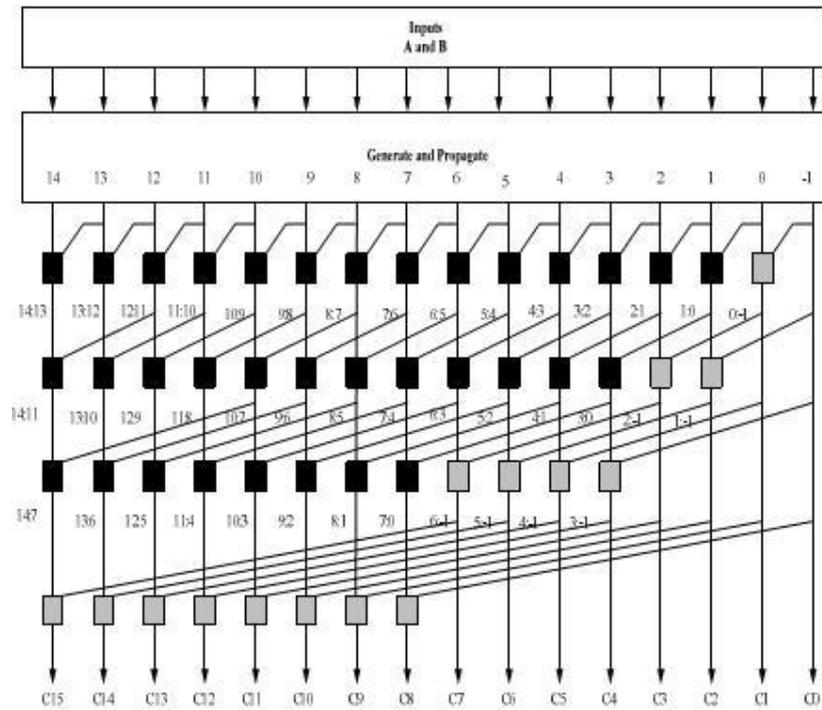
16 Bit Spanning Tree Adder

In this adder initially pre computation stage we are doing propagation and generation operations. After that stage1 used 6 black cells. Stage2 used 3 black cells. Stage3 used. 2 black cells and one gray cell.stage4 used two gray cells and one black cell.stage5 used one gray cell then applies ripple carry adder operation.



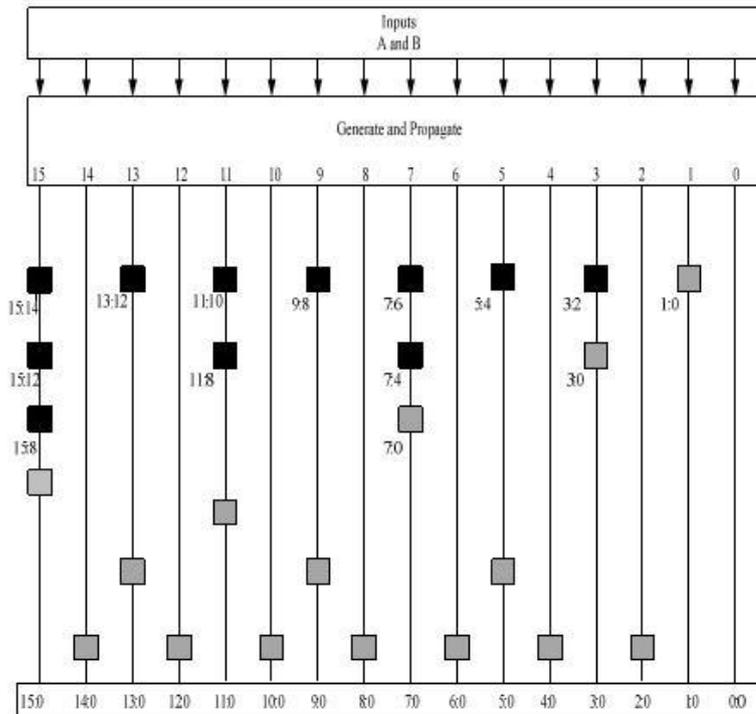
16 Bit Kogge Stone Adder

In this adder initially pre computation stage we are doing propagation and generation operations. After that stage1 used 14 black cells and one gray cell. Stage2 used 12 black cells and two gray cells. Stage3 used. 8 black cells and 4 gray cell. Stage4 used 8 gray cells then final computing operation.



16 Bit Brent Kung Adder

In this adder initially pre computation stage we are doing propagation and generation operations. After that stage1 used 7black cells and one gray cell. Stage2 used 3 black cells and one gray cell. Stage3 used. 1 black cells and one gray cell.stage4 used one gray cell.stage5 used one gray cell.stage6 Used 3 gray cells. Stage7 used 7 gray cells then final computing operation.

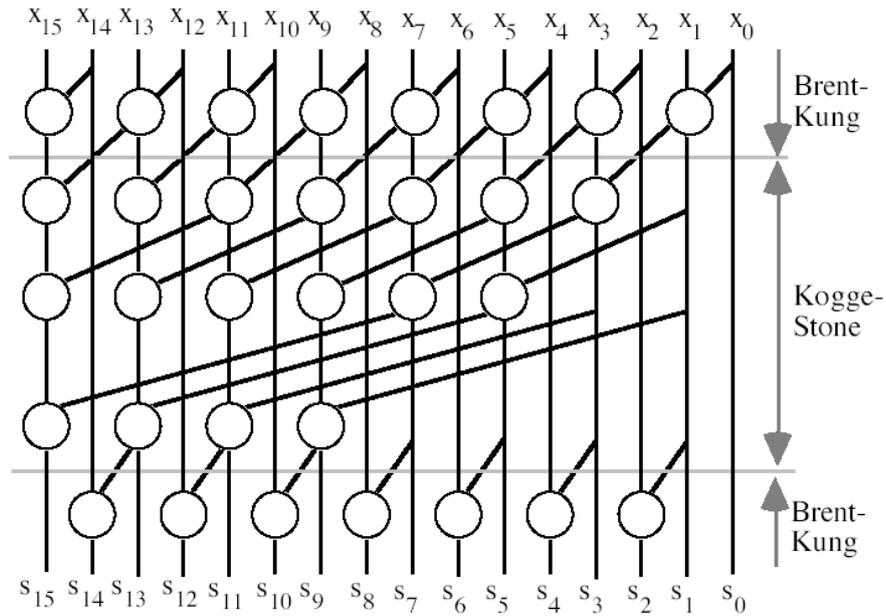


Hybrid Parallel Prefix Adder

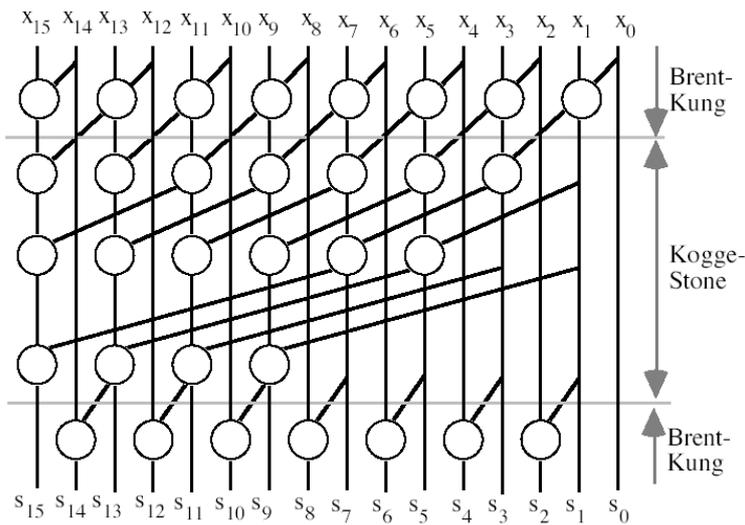
We are combining Kogge stone adder and brentkung adder. Kogge stone adder having four levels and 49 cells and brentkung adder having six levels and 29 cells in our proposed hybrid adder five levels and 32 cells in first level same as brent kung adder. Second, third ,four levels are same as Kogge stone

adder and finally fifth level belongs to brent kung adder then final output giving to final computing operation

Hybrid: Five levels, 32 cells



Hybrid: Five levels, 32 cells



SIMULATION WAVEFORMS AND SCHEMATIC DIAGRAMS

Brent Kung adder



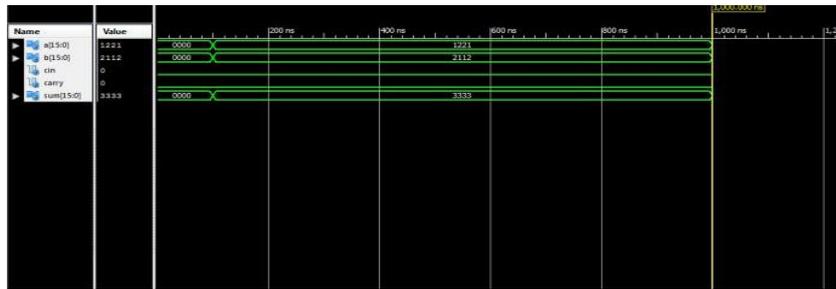
Kogge Stone Adder



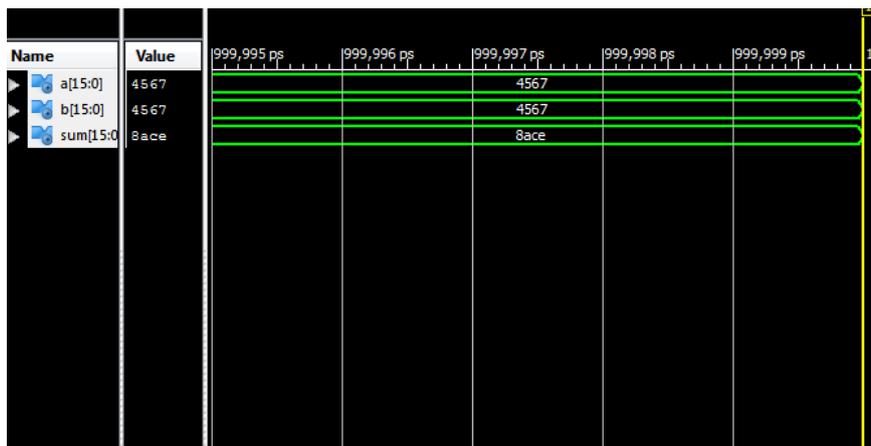
Spanning Tree Adder



Sparse Kogge-Stone Adder



Hybrid Parallel Prefix Adder



CONCLUSION

In this studied about prefix adder’s concept. Here we have four types of prefix adders. Those are Brent Kung adder, Kogge stone adder, sparse Kogge stone adder, spanning tree adder And we are generated a new hybrid adder using brentkung adder and Kogge stone adder. Five adders are simulated and synthesize using Xilinx 13.2 and hardware kit Spartan 3e.

REFERENCES

- [1] David H.K.Hoe, Chris Martinez and Sri JyothsnaVundavalli”, Design and Characterization of Parallel Prefix Adders using FPGAs”, 2011 IEEE 43rd South-eastern Symposium in pp. 168-172, 2011.
- [2] N. H. E. Weste and D. Harris, CMOS VLSI Design, 4th edition, Pearson–Addison-Wesley, 2011.

- [3] R. P. Brent and H. T. Kung, “A regular layout for parallel adders,” IEEE Trans. Computer., vol. C-31, pp. 260-264, 1982.
- [4] D. Harris, Taxonomy of Parallel Prefix Networks,” in Proc. 37th Asilomar Conf. Signals Systems and Computers, pp. 2213–7, 2003.
- [5] P. M. Kogge and H. S. Stone, “A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations,” IEEE Trans. on Computers, Vol. C-22, No 8, August 1973.
- [6] D.Gizopoulos, M. Psarakis, A. Paschalis, and Y. Zorian, “Easily Testable Cellular Carry Look ahead Adders,” Journal of Electronic Testing: Theory and Applications 19, 285-298, 2003.
- [7] T. Lynch and E. E. Swartzlander, “A Spanning Tree Carry Look ahead Adder,” IEEE Trans. on Computers, vol. 41, no. 8, pp. 931-939, Aug. 1992.
- [8] Beaumont-Smith, A, Cheng-Chew Lim,”Parallel prefix adder design”, Computer Arithmetic, 2001. Proceedings. 15th IEEE Symposium, pp. 218 – 225, 2001.M. Young, the Technical Writer's Handbook. Mill Valley, CA: University Science, 1989
- [9] K.Vitoroulis and A.J. Al-Khalili, “Performance of Parallel Prefix Adders Implemented with FPGA technology,” IEEE Northeast Workshop on Circuits and Systems, pp. 498-501, Aug. 2007. 172.
- [10] S. Xing and W. W. H. Yu, “FPGA Adders: Performance Evaluation and Optimal Design,” IEEE Design & Test of Computers, vol. 15, no. 1, pp. 24-29, Jan. 1998

AUTHORS' BIOGRAPHY

TANVEERFATIMA has completed her B.Tech in Electronics and Communication Engineering from Teegala Krishna Reddy Engineering College, J.N.T.U.H affiliated in 2013. She is pursuing her M.Tech in VLSI System Design from Siddhartha Institute of Technology and Sciences, J.N.T.U.H affiliated college.

SREE UMA is an Associate Professor at, Siddhartha Institute of Technology and Sciences, Hyderabad. She received her Master’s degree in V.L.S.I System Design from J.N.T.U.H affiliated college.