International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 3 Issue 9, September- 2014

Design and Characterization of Efficient Parallel Prefix Adders using FPGAs

Vanga Mahesh (M.Tech),
KITS, Warangal, Telangana,
India.

R. Nirmala Devi. M.Tech, (Ph.D)
Associate Professor ,
KITS, Warangal, Telangana,
India.

***Abstract*—The binary adder is that the essential part in most digital circuit styles as well as digital signal processors (DSP) and microchip knowledge path units. As such, in depth analysis continues to be centered on raising the ability delay performance of the adder. The Ripple carry adder and carry skip adder's output of every stage depends on the previous carry .But once coming back to hold tree adders (Parallel prefix adders), It generates the carry signals in O(log n)time. and are better-known to own the simplest performance in VLSI styles. The carry-tree adders (the Kogge-Stone, Sparse Kogge-Stone, and spanning tree adder) are compared with straightforward Ripple Carry Adder (RCA) and Carry Skip Adder (CSA) exploitation high performance logic instrument. Due to the presence of a quick carry-chain, the RCA styles exhibit higher delay performance up to 128 bits. whereas carry-tree adders have a speed of advantage over the RCA as bit widths approach 256**

Key Words: Parallel Prefix adders, kogge stone adder, Sparse kogge stone adder, Spanning Tree adder, Logic analyzer.

1. INTRODUCTION

Binary addition is fundamental operation in most of the digital circuits. There are so many adders in the digital design. The selection of adder depends on its performance parameters. Adders are important elements in microprocessors, digital signal processors.ALU and in floating point arithmetic units. and memory addressing ,in booth multipliers .they are also used in real time signal processing like signal processing, image processing etc. for human beings arithmetic calculations are easy to calculate when they are decimals i.e. base ten. But they became pragmatic if binary numbers are given. Therefore binary addition is essential any improvement in binary addition can improve the performance of system. The fast and accuracy of system depends mainly on adder performance.

In this paper designing and implementation of various parallel prefix adders on FPGA are described. Parallel Prefix Adders are also known as Carry Tree Adders. Parallel prefix adders are designed from carry look ahead adder as a base. Parallel prefix adders consist of three stages similar to CLA. Figure 1 shows the PPA structure.

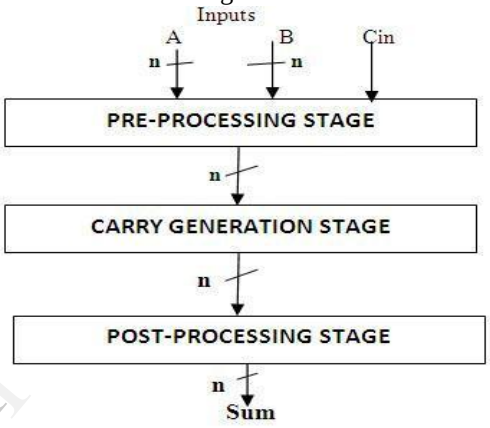

Figure 1.1 Block diagram of PPA

The parallel prefix adder employs three stages in pre-processing stage the generation of Propagate and Generate signals is carried out. The calculation of Generate (Gi) and Propagate (Pi) are calculated when the inputs A, B are given. As follows

Gi=Ai AND Bi

Pi=Ai XOR Bi

Gi indicates whether the Carry is generated from that bit. Pi indicates whether Carry is propagated from that bit.

In carry generation stage of PPA, prefix graphs can be used to describe the tree structure. Here the tree structure consists of grey cells, black cells, and buffers. In carry generation stage when two pairs of generate and propagate signals (Gm, Pm), (Gn, Pn) are given as inputs to the carry generation stage. It computes a pair of group generates and group propagate signals (Gm: n, Pm: n) which are calculated as follows

Gm: n=Gm+ (Pm.Gn)
Pm: n=Pm. Pn

The black cell computes both generate and propagate signals as output. It uses two and gates and or gate. The

grey cell computes the generate signal only. It uses only and gate, or gate.

In post processing stage simple adder to generate the sum, Sum and carry out are calculated in post processing stage as follows

Si=Pi XOR Ci-1

Cout=Gn-1 XOR (Pn-1 AND Gn-2)

If Cout is not required it can be neglected.

2. CARRY TREE ADDER STRUCTURES

Parallel prefix adders also known as carry tree adders

They pre-compute propagate and generate signals. These signals are combined using fundamental carry operator (fco).

(g1, p1) o (g2, p2) = (g1+g2.p1, p1.p2)

Due to associative law of the fundamental carry operator these operators can be combined in different ways to form various adder structures. For example 4 bit carry look ahead generator is given by

C4= (g4, p4) o [(g3, p3) o [(g2, p2) o (g1, p1)]]

Now in parallel prefix adders allow parallel operation resulting in more efficient tree structure for this 4 bit example.

C4= [(g4, p4) o (g3, p3)] o [(g2, p2) o (g1, p1)]

It is a key advantage of tree structured adders is that the critical path due to carry delay is of order log2N for N bit wide adder. So the arrangement of the prefix network gives rise to various families of adders. For this study the focus is on Kogge-Stone, Sparse Kogge stone, and Spanning Tree adders.. Here we designate black cell as BC and grey cell as GC.

2.1. Kogge-Stone adder

Kogge-Stone adder is one among the parallel prefix adders. This has regular layout which makes them favoured adder in electronic technology. It has the minimum fan-out. A 16 bit Kogge stone adder is shown in the figure 2.

The maximum fan-out is 2 in all the logic levels for all width Kogge-stone prefix trees. The key of building any prefix tree is to implement the equation according to the specific features and apply the rules above described in the previous section. The number of stages for a Kogge stone adder is calculated by log2 power N. It consists of 34 BC's and 15 GC's and buffers are given.

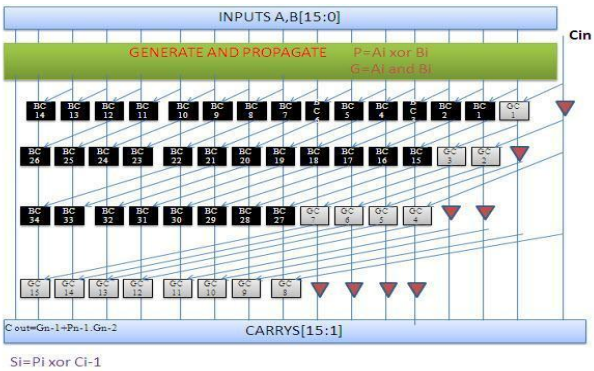

Figure 2.1 Block Diagram of 16 bit Kogge Stone Adder

2.2. Sparse Kogge-Stone adder

The Sparse Kogge stone adder consists of several small ripple carry adders on its lower part, a carry tree is on its upper part. It terminates with ripple carry adders. Number of carries generated is less in this adder compared to Kogge stone adder. The function of grey cells and black cells is same as discussed in previous sections. Figure .3 shows the block diagram of Sparse Kogge Stone adder.

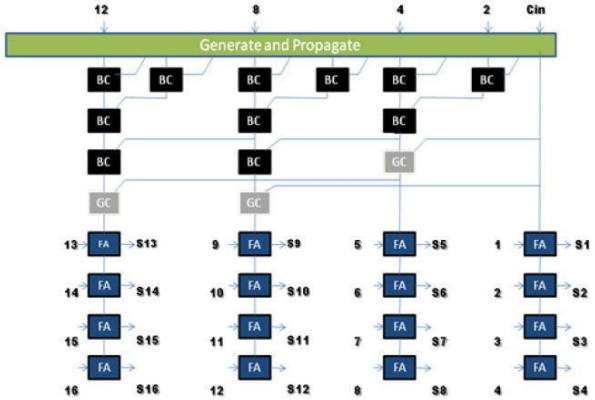

Figure 2.2 Block Diagram of 16 bit Sparse Kogge Stone Adder

2.3. Spanning Tree adder

Another carry-tree adder known as the spanning tree carry look ahead (CLA) adder is also examined. Like the sparse kogge-Stone adder, this design terminates with a 4- bit RCA. As the FPGA uses a fast carry-chain for the RCA, it is interesting to compare the performance of this adder with the sparse kogge-Stone and regular kogge-Stone adders. It also uses the black cells and gray cells and full adder blocks like sparse kogge stone adders but the difference is the interconnection between them.

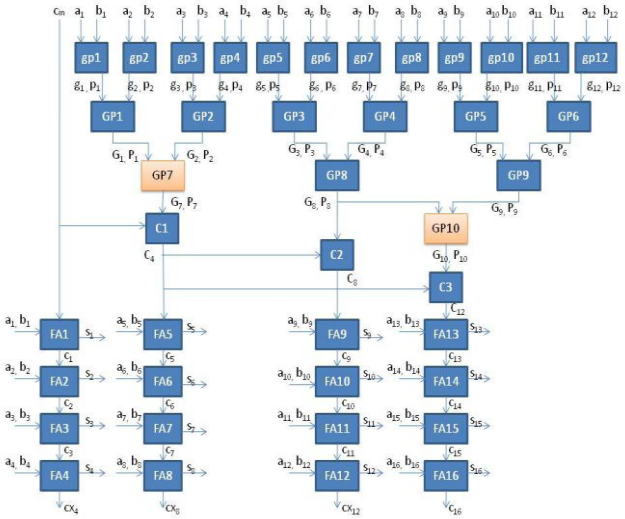

Figure 2.3 Block Diagram of 16 bit Spanning Tree Adder

3. RELATED WORK

We compared the design of the ripple carry adder with carry look ahead adder and different Parallel prefix trees. The Previous authors considered several Parallel prefix adders implemented on Xilinx vertex 5 FPGA.it is found that ripple carry adder performs better than carry tree designs because RCA can take advantage of fast carry chain on the FPGA, H.K .Hoe, Chris Martinez and Jyothsna Vundavalli concluded Kogge stone adder is best in terms of delay. But it takes larger area..Now in this paper we focus on carry tree adders implemented on Xilinx Spartan 3E FPGA. Here we design different carry tree adders and compared with Ripple carry adder in terms of delay. We also compare with Kogge-Stone Adder in terms of area by counting of number of LUT's and Slices.

4. METHODOLOGY

. The adders to be studied were designed with varied bit width bits and they are coded in VERILOG. The verification of the adders was verified by using Model-sim Simulator. The Xilinx ISE 13.2 software was used to synthesize the designs onto Spartan 3E FPGA .By using the Generate and Propagate and by BC and GC we are able to develop the Carry trees. It is found that the Kogge Stone Prefix trees provide better delay performance for higher order bits. We seen area is high.

5. SIMULATION AND SYNTHESIS AND FPGA RESULT

The Ripple carry adders, Carry look ahead adder, and Kogge-Stone adder, Sparse Kogge Stone Adder and Spanning Tree adder are simulated and synthesised written in verilog using model-sim and Xilinx ISE tools. We noticed that parallel prefix adders are faster than the ripple carry adder. The results of different parallel prefix adders are as given below.

5.1 SYNTHESIS RESULT

For Synthesis,

1. In the Design panel, select Implementation from the Design View drop-down list.
2. In the Hierarchy pane. select the top module Image.
3. In the Processes pane, double-click Synthesize.

This device utilization includes the following.

- Logic Utilization
- Logic Distribution
- Total Gate count for the Design

The device utilization summery is shown above in which its gives the details of number of devices used from the available devices and also represented in %.

Kogge Stone Adder

Kogge_Stone Project Status			
Project File:	kogge16.xise	**Parser Errors:**	No Errors
Module Name:	Kogge_Stone	**Implementation State:**	Synthesized
Target Device:	xc5vlx110t-2ff1136	**• Errors:**	No Errors
Product Version:	ISE 13.2	**• Warnings:**	4 Warnings (0 new)
Design Goal:	Balanced	**• Routing Results:**	
Design Strategy:	Xilinx Default (unlocked)	**• Timing Constraints:**	
Environment:	System Settings	**• Final Timing Score:**	

Device Utilization Summary (estimated values) [-]			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice LUTs	71	69120	0%
Number of fully used LUT-FF pairs	0	71	0%
Number of bonded IOBs	50	640	7%

Figure 5.1: device utilization summary of kogge stone adder

The kogge stone adder takes 71 look up tables out of 69120 and 50 bounded IOBs out of 640.No flip flops utilized in this adder.

Sparse Kogge Stone Adder

sparsekogge Project Status (07/17/2013 - 17:12:06)			
Project File:	sparsekogge.xise	**Parser Errors:**	No Errors
Module Name:	sparsekogge	**Implementation State:**	Synthesized
Target Device:	xc5vlx110t-2ff1136	**• Errors:**	No Errors
Product Version:	ISE 13.2	**• Warnings:**	4 Warnings (0 new)
Design Goal:	Balanced	**• Routing Results:**	
Design Strategy:	Xilinx Default (unlocked)	**• Timing Constraints:**	
Environment:	System Settings	**• Final Timing Score:**	

Device Utilization Summary (estimated values) [-]			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice LUTs	28	69120	0%
Number of fully used LUT-FF pairs	0	28	0%
Number of bonded IOBs	50	640	7%

Figure 5.2: device utilization summary of sparse kogge stone adder

The sparse kogge adder takes 28 look up tables out of 69120 and 50 bounded IOBs out of 640.No flip flops utilized in this adder.

Spanning tree Adder

spanningtree Project Status (07/16/2013 - 16:57:45)			
Project File:	spanningtreefinal.xise	**Parser Errors:**	No Errors
Module Name:	spanningtree	**Implementation State:**	Synthesized
Target Device:	xc5vlx110t-2ff1136	**• Errors:**	No Errors
Product Version:	ISE 13.2	**• Warnings:**	14 Warnings (0 new)
Design Goal:	Balanced	**• Routing Results:**	
Design Strategy:	Xilinx Default (unlocked)	**• Timing Constraints:**	
Environment:	System Settings	**• Final Timing Score:**	

Device Utilization Summary (estimated values) [-]			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice LUTs	30	69120	0%
Number of fully used LUT-FF pairs	0	30	0%
Number of bonded IOBs	49	640	7%

Figure 5.3: device utilization summary of spanning tree adder

The spanning tree adder takes 30 look up tables out of 69120 and 49 bounded IOBs out of 640.No flip flops utilized in this adder.

5.2 SIMULATION RESULTS

ISim provides a complete, full-featured HDL simulator integrated within ISE. HDL simulation now can be an even more fundamental step within your design flow with the tight integration of the ISim within your design environment.

The Xilinx® ISE Simulator (ISim) is a Hardware Description Language (HDL) simulatorthat enables you to perform functional (behavioral) and timing simulations for VHDL,

verilog and mixed language designs. The Xilinx® ISE Design Suite provides an integrated flow with the ISE Simulator (ISim)that allows simulations to be launched directly from the Project Navigator (ISE). All simulation commands that prepare the ISim simulation are generated by ISE Project navigator and automatically run in the background when simulating a design using thisflow

Kogge Stone Adder

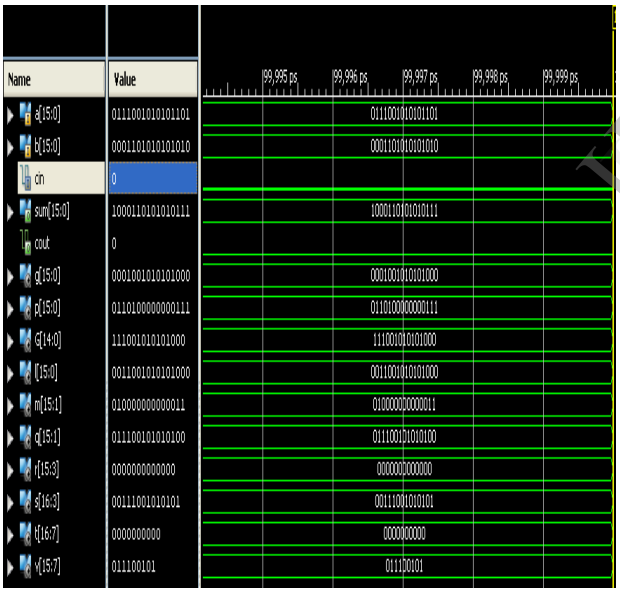

Figure 5.4: simulation result of kogge stone adder

From the above figure 5.15, two inputs a, b (16 bit each) and input carry (cin) is applied to the kogge stone adder block, and then output sum (16 bit) and output carry (cout) are observed.

For kogge stone adder, inputs are given as a=0111001010101101, b=0001101010101010 and cin=0 then sum=1000110101010111 and cout=0 are obtained.

Sparse Kogge Stone Adder

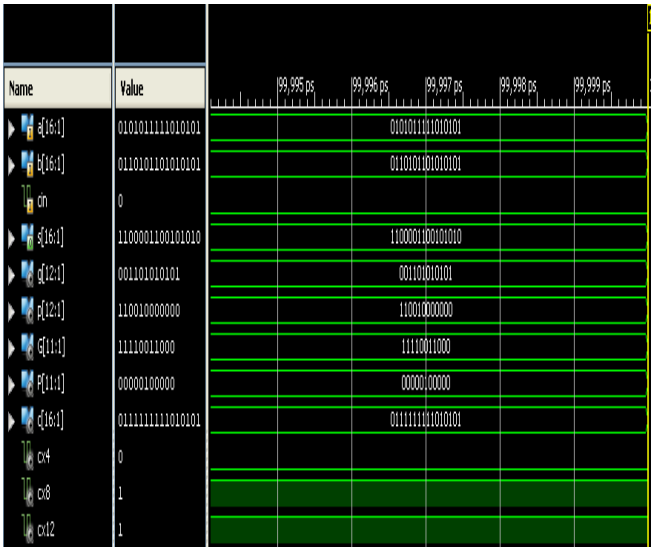

Figure 5.5: Simulation result of sparse kogge stone adder

From the above figure 5.15, two inputs a, b (16 bit each) and input carry (cin) is applied to the sparse kogge stone adder block, then output sum(16 bit) and output carry(cout) are observed.

For sparse kogge stone adder, inputs are given as a=0101011111010101, b=0110101101010101 and cin=0 then sum=1100001100101010 and cout=0 are obtained.

Spanning tree Adder

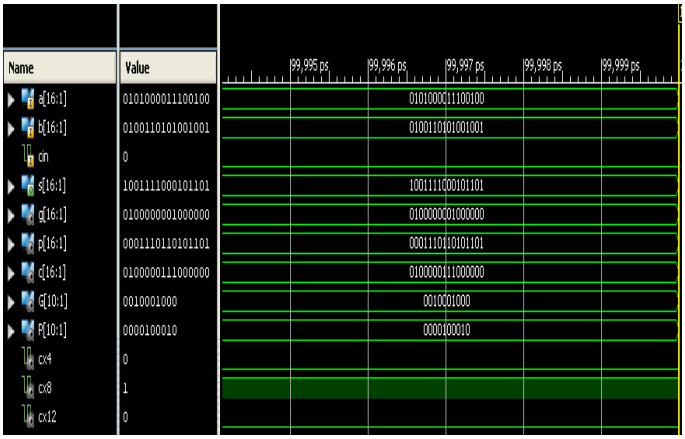

Figure 5.6: simulation result of spanning tree adder

From the above figure 5.15, two inputs a, b (16 bit each) and input carry (cin) is applied to the spanning tree adder block, then output sum(16 bit) and output carry(cout) are observed.

For spanning tree adder, inputs are given as a=0101000011100100, b=0100110101001001 and cin=0 then sum=1001111000101101 and cout=0 are obtained.

5.3 FPGA RESULT

A field-programmable gate array (FPGA) is an integrated circuit designed to be configured by a customer or a designer after manufacturing—hence "field-programmable". FPGA contains a two dimensional arrays of logic blocks and interconnections between logic blocks. Both the logic blocks and interconnects are programmable. Logic blocks are programmed to implement a desired function and the interconnections are programmed using the switch boxes to connect the logic blocks.To be more clear, if we want to implement a complex design (CPU for instance), then the design is divided into small sub functions and each sub function is implemented using one logic block. FPGAs contain programmable logic components called "logic blocks", and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together"—somewhat like many (changeable) logic gates that can be inter-wired in (many) different configurations. Logic blocks can be configured to perform complex combinational functions, or merely simple logic gates like AND and XOR. In most FPGAs, the logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory.

Kogge Stone Adder

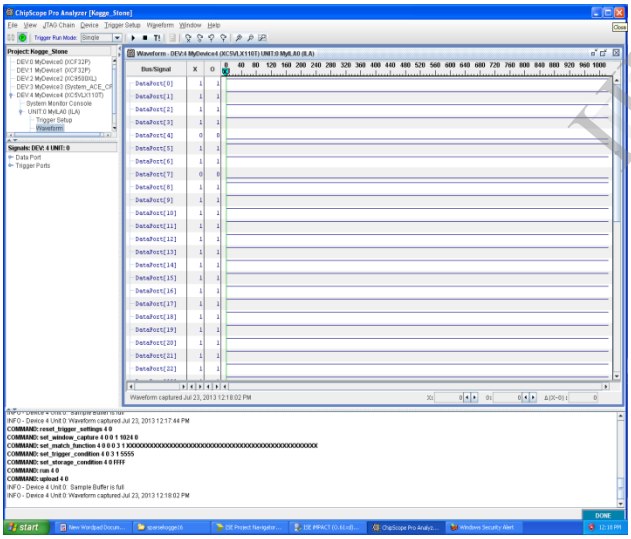

Figure 5.7: simulation result of kogge stone adder

From the above figure 5.41, two inputs a, b (16 bit each) and input carry (cin) is applied to the kogge stone adder block, and then output sum (16 bit) and output carry (cout) are observed.

So the inputs are given as a=0111001010101101, b=0001101010101010 and cin=0 then sum=1000110101010111 and cout=0 are obtained.

Sparse Kogge Stone Adder

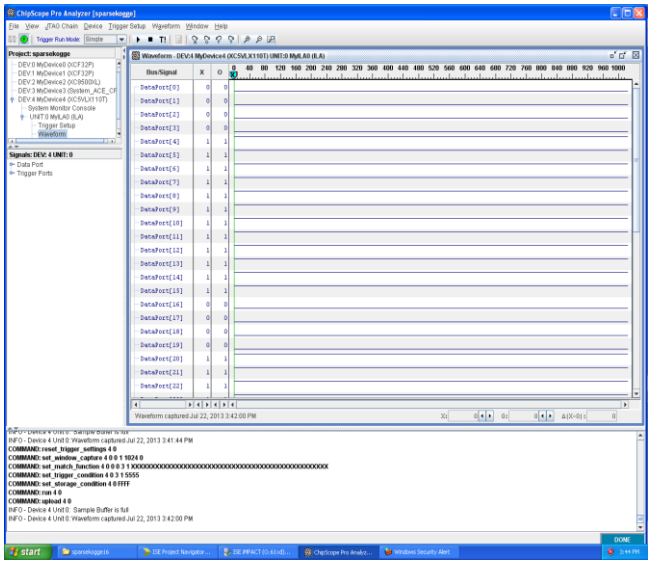

Figure 5.8: simulation result of sparse kogge stone adder

From the above figure 5.42, two inputs a, b (16 bit each) and input carry (cin) is applied to the sparse kogge stone adder block, then output sum(16 bit) and output carry(cout) are observed.

So the inputs are given as a=0101011111010101, b=0110101101010101 and cin=0 then sum=1100001100101010 and cout=0 are obtained.

Spanning tree Adder

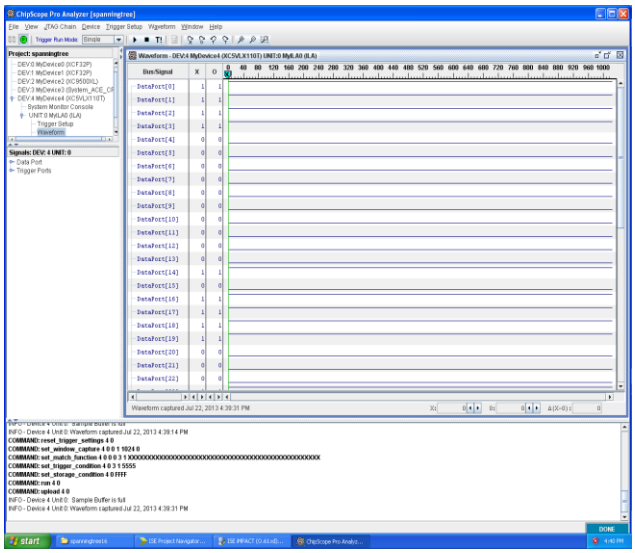

Figure 5.9: simulation result of spanning tree adder

From the above figure 5.43, two inputs a, b (16 bit each) and input carry (cin) is applied to the spanning tree adder block, then output sum(16 bit) and output carry(cout) are observed.

So the inputs are given as a=0101000011100100, b=0100110101001001 and cin=0 then sum=1001111000101101 and cout=0 are obtained.

Table 5.1: comparison of all adders delay

S.no.	Adder Name (16 bit)	Xilinx ISE 13.2 Tool delay (in ns)	From Ref[1] Delay (in ns)	Logic Analyzer Delay (in ns)	Power (in Watts)	Device Utilization (LUTs, IOBs) (69120, 640)
1	Ripple carry adder	3.853	2.578	2.437	1.211	5,13
2	Kogge stone adder	6.688	6.286	5.048	1.220	71,50
3	Sparse kogge stone adder	8.014	-------	5.151	1.179	28,50
4	Spanning tree adder	6.667	-------	5.142	1.179	30,49

The comparison is made for all adders in the project.

6. CONCLUSION

Both measured and simulation results from this study have shown that parallel-prefix adders don't seem to be as effective as the simple ripple-carry adder at low to moderate bit widths. This is not sudden because the Xilinx FPGA incorporates a quick carry chain that optimizes the performance of the ripple carry adder. However, contrary to alternative studies, we have indications that the carry-tree adders eventually surpass the performance of the linear adder styles at high bit-widths, expected to be within the 128 to 256 bit vary. this is often necessary for large adders utilized in exactness arithmetic and cryptographic applications wherever the addition of numbers on the order of 1000 bits isn't uncommon.
as a result of the adder is commonly the essential part that determines to a large half the cycle time and power dissipation for several digital signal process and cryptanalytic implementations, it might be worthy for future FPGA designs to incorporate associate degree optimized carry path to change tree based adder styles to be optimized for place and routing. This would improve their performance just like what's found for the RCA. we have a tendency to decide to explore potential FPGA architectures that would implement a "fast-tree chain" and investigate the potential trade-offs concerned. The inherent redundancy of the Kogge-Stone carry-tree structure and its implications for fault tolerance in FPGA styles is being studied. The testability and potential fault tolerant options of the spanning tree adder also are topics for future analysis.

ACKNOWLEDGEMENT

I Vanga Mahesh would like to thank R.Nirmaladevi M.Tech (Ph.D), who had been guided throughout the project and supporting me in giving technical ideas about the paper and motivating to complete the work successfully.

REFERENCES

[1] David H. K. Hoe, Chris Martinez and Sri JyothsnaVundavalli,"Design and Characterization of Parallel Prefix Adders using FPGAs", 2011 IEEE 43rd Southeastern Symposium in pp. 168-172, 2011.
[2] Furber, S.B. ,Liu, J.," A novel area-efficient binary adder", Signals, Systems and Computers, 2000. Conference Record of the Thirty-Fourth Asilomar Conference on (Volume:1),pp. 119 - 123 vol.1, Oct. 29 2000-Nov. 1 2000.
[3] Han, Tackdon, Carlson, D.A.,"Fast area-efficient VLSI adders", Computer Arithmetic (ARITH), 1987 IEEE 8th Symposium,pp. 49 – 56, 18-21 May 1987.
[4] D. Gizopoulos, M. Psarakis, A. Paschalis, and Y. Zorian, "Easily Testable Cellular Carry Lookahead Adders," *Journal of Electronic Testing: Theory and Applications 19*, 285-298, 2003.
[5] 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.
[6] M. Becvar and P. Stukjunger, "Fixed-Point Arithmetic in FPGA," *ActaPolytechnica*, vol. 45, no. 2, pp. 67- 72, 2005.
[7] 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.
[8] N. H. E. Weste and D. Harris, *CMOS VLSI Design*, 4th edition, Pearson–Addison-Wesley, 2011.
[9] R. P. Brent and H. T. Kung, "A regular layout for parallel adders," *IEEE Trans. Comput.*, vol. C-31, pp. 260-264, 1982.
[10] D. Harris, "A Taxonomy of Parallel Prefix Networks," in *Proc. 37th Asilomar Conf. Signals Systems and Computers*, pp. 2213–7, 2003.
[11] 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.
[12] P. Ndai, S. Lu, D.Somesekhar, and K. Roy, "Fine-Grained Redundancy in Adders," *Int. Symp.on Quality Electronic Design*, pp. 317-321, March 2007.
[13] T. Lynch and E. E. Swartzlander, "A Spanning Tree Carry Lookahead Adder," *IEEE Trans. on Computers*, vol. 41, no. 8, pp. 931-939, Aug. 1992.
[14] 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.
[15] Beaumont-Smith, A, Cheng-Chew Lim ,"Parallel prefix adder design", Computer Arithmetic, 2001. Proceedings. 15th IEEE Symposium,pp. 218 – 225,2001.
[16] Martinez, C.D., Bollepalli, L.P.D. Hoe, D.H.K.,"A fault tolerant parallel-prefix adder for VLSI and FPGA design", System Theory (SSST), 2012 44th Southeastern Symposium,pp. 121 – 125, 11-13 March 2012.