International Journal of
Scientific Engineering and Technology Research

ISSN 2319-8885
Vol.02,Issue.06,
July-2013,
Pages:467-479

www.semargroups.org

Design and Characterization of Sparse Kogge Stone Parallel Prefix Adder Using FPGA

E.SREENIVASA GOUD[1], P.C.PRAVEEN KUMAR[2]
[1]Research Scholar, ECE Dept, Kottam College of Engineering, Chinnatekur, Kurnool, AP-INDIA,
E-mail:esgoudies@gmail.com.
[2]Assoc Prof, ECE Dept, Kottam College of Engineering, Chinnatekur, Kurnool, AP-INDIA,
E-mail:chenna05@gmail.com.

Abstract: The binary adder is the critical element in most digital circuit designs including digital signal processors (DSP) and microprocessor data path units. Such, as the extensive research continues to be focused on improving the power-delay and then performance of the adder. In the VLSI implementations and the parallel-prefix adders (also known as carry-tree adders) are known to have the best performance. However, this performance advantage does not translate directly into FPGA implementations due to constraints on logic block configurations and then routing overhead. In this paper we can investigates three types of carry-tree adders (the Kogge-Stone, sparse Kogge-Stone, and then spanning tree adder) and compares them to the simple Ripple Carry Adder (RCA). These designs of varied bit-widths were implemented on a Xilinx Virtex5 FPGA and delay values were taken from static timing analysis of synthesis results obtained from Xilinx ISE design suite 10.1. Due to the presence of the fast carry-chain, when the RCA designs are we can exhibit better delay performance up to 64 bits. The carry-tree adders have a speed advantage over the RCA as bit widths approach 256.

Keywords: Ripple Carry Adder (RCA) and Carry Skip Adder (CSA), FPGA implementations, power-delay performance, carry tree Adders, VLSI designs, digital signal processors (DSP).

I. INTRODUCTION

The binary adder is the critical element in most digital circuit designs including digital signal processors (DSP) and microprocessor units. And such as, extensive research continues to be focused on improving the power-delay performance of the adders. VLSI implementations, parallel-prefix adders are known to have the best performance. Reconfigurable logic such as Field Programmable Gate Arrays (FPGAs) has been gaining in popularity in recent years because it offers improved performance in terms of speed and power over DSP-based and microprocessor-based solutions for many practical designs involving mobile DSP and telecommunications applications and a significant reduction in development time and cost over Application Specific Integrated Circuit (ASIC) designs.

The power advantage is especially important with the growing popularity of mobile and portable electronics, which make extensive use of DSP functions. However, because of the structure of the configurable logic and routing resources in FPGAs, parallel-prefix adders will have a different performance than VLSI implementations. In particular, most modern FPGAs employ a fast-carry chain Which optimizes the carry path for the simple Ripple Carry Adder (RCA). In this paper, the practical issues involved in designing and implementing tree-based adders on FPGAs are this work was supported in part by NSFLSAMP and UT-System STARS awards. The FPGA ISE synthesis software was supplied by the Xilinx University program described.

An efficient testing strategy for evaluating the performance of the adders is discussed. Then the several tree-based adder structures are implemented and characterized on a FPGA and compared with the Ripple Carry Adder (RCA) and the Carry Skip Adder (CSA). Finally, some conclusions and suggestions for improving FPGA designs to enable better tree-based adder performance are given. Parallel-prefix structures are found to be common in high performance adders because of the delay is logarithmically proportional to the adder width. Such structures can usually be divided into three stages

1. Pre-computation

2. Prefix tree

3. Post-computation

II. PARALLEL-PREFIX ADDITION

When the binary adder is the critical element in most digital circuit designs including digital signal processors (DSP) and then microprocessor data path units. Such, as the extensive research continues to be focused on improving the power delay performance of the adder. In the VLSI implementations, and then parallel-prefix adders are known to have the best performance. Reconfigurable logic such as Field Programmable Gate Arrays (FPGAs) has been gaining in popularity in recent years because it offers improved performance in terms of speed and power over DSP-based and microprocessor-based solutions for many practical designs involving mobile DSP and telecommunications applications and a significant reduction in development time and cost over Application Specific Integrated Circuit (ASIC) designs. The power advantage is especially important with the growing popularity of mobile and portable electronics, which make extensive use of DSP functions.

However, because of the structure of the configurable logic and routing resources in FPGAs, parallel-prefix adders will have a different performance than VLSI implementations. In particular, most modern FPGAs employ a fast-carry chain which optimizes the carry path for the simple Ripple Carry Adder (RCA). In this paper, the practical issues involved in designing and implementing tree-based adders on the FPGAs. An efficient testing strategy for evaluating the performance of these adders is discussed. Several tree-based adder structures are implemented and characterized on a FPGA and compared with the Ripple Carry Adder (RCA) and the Carry Skip Adder (CSA).

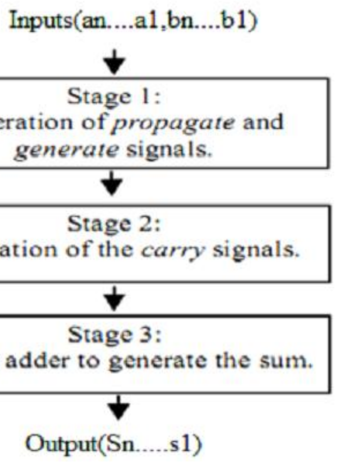

Inputs(an....a1,bn....b1)
Stage 1:
Generation of propagate and generate signals.
Stage 2:
Generation of the carry signals.
Stage 3:
Simple adder to generate the sum.
Output(Sn.....s1)

Fig.1. Block Diagram of Prefix addition.

Finally, some conclusions and suggestions for improving FPGA designs to enable better tree-based adder performance are given. The problems are involved in FPGA implementation are investigated and the possible FPGA architecture which can make the Carry Tree Adder to provide high performance over the Simple adder it can be explored. Then the possible trade-offs like area, power, delay, interconnect count and fan-out involved in the adders are examined.

There are three stages the addition it consists of the following computations:

• Pre-computation:

$$G_{m:n}=A_n \text{ and } B_n,\ G_0=c_{in};\ P_{m:n}=A_n \text{ xor } B_n,\ P_0=0; \qquad \textbf{(1)}$$

• Prefix-computation:

$$(G_m, P_m) \text{ o } (G_n, P_n) = (G_{n:k} + P_{n:k}\cdot G_{k-1:n},\ P_{n:k}\cdot P_{k-1:j}) \text{ (or) } G_{m:n}=G_{n:k}+P_{n:k}\cdot G_{k-1:n}\ P_{m:n}=P_{n:k}\cdot P_{k-1} \qquad \textbf{(2)}$$

• Post-computation:

$$S_n=P_n \text{ xor } G_{n-1:0} \qquad \textbf{(3)}$$

III. SPARSE KOGGE-STONE ADDER GENERATOR

This generates Verilog code for adders with large numbers of bits. While a complete adder would produce the output of all bits, this just outputs a series of carry bits at fixed intervals. These can be used as the carry-in bits for a series of smaller adders. This is useful in particular for FPGAs, where small ripple-carry adders can be much faster than general-purpose logic thanks to fast connections between neighbouring slices. This allows a large adder to be composed of many smaller adders by generating the intermediate carries quickly.

A. Options

Bits in adder: 128

Bits between carry outputs: 16

Background: When we add numbers on paper, we would do this in our normal base 10 counting system by adding together the digits in the smallest place first, then moving up until we've reached the largest digit. If any pair of digits added together reaches 10, the 10 gets taken out and added to the next highest place, where it's equivalent to a 1 digit at that magnitude.

$$\begin{array}{rr} & 1\ \overset{1}{2}\ \overset{1}{3}\ \overset{1}{4}\ 5 \\ + & 6\ 7\ 8\ 9 \\ \hline = & 1\ 9\ 1\ 3\ 4 \end{array}$$

When we add in base 10, we add pairs of digits and carry up to the next place.

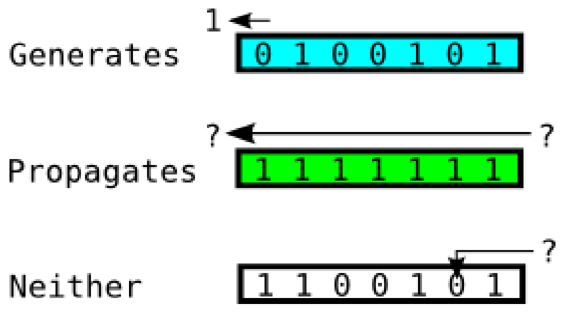
In binary, we do exactly the same thing. In binary though, we only have 0s and 1s in each place. We add together the digits and any carry from the previous position, and if put together they reach 2, the 2 is taken out and carried to the next place up, where it's equivalent to a 1 digit at that magnitude. We get a very simple table which shows what the value and carry are for any combination:

First Digit	Second Digit	Carry from previous	Total	Carry to next place	Result in this place
0	0	0	0	0	0
0	1	0	1	0	1
1	0	0	1	0	1
1	1	0	2	1	0
0	0	1	1	0	1
0	1	1	2	1	0
1	0	1	2	1	0
1	1	1	3	1	1

This table is easily implemented with a few digital gates, and by stringing together a whole series of these from the smallest to the largest place, we get a "ripple carry" adder, which adds exactly like we would on paper. Gates, however, take a certain amount of time to produce a stable output. While on a much smaller time scale than we're used to dealing with, gate delay is what constrains the speed a processor or IC can run at. The ripple carry adder is so named because the carry from the smallest bit affects the output and carry of the bit one place up, and as each place is worked out the correct result appears as a "ripple" from smallest to largest bit.

When measuring the time it takes to do a calculation using gates, we have to add together the gate delays and find the longest route through the gates to a result we depend on. Since in the adder the output of each bit depends on the carry from the next bit down, we have a long chain where every bit, including the largest, is affected by the input to the smallest bit. The longest path here is from the input to the smallest place to the output of the largest place of the result. This is proportional to the number of bits - doubling the number of bits will double the time it takes to add the numbers together. Ripple-carry adders for large numbers can take a long time to stabilize as a result.

To get around this problem and calculate faster, we can use the idea that the carry behavior of any stretch can be described by "propagate" and "generate" flags. A stretch in the addition, which could be just a single place or a range from one to another, can be described by these behaviors.



Generate blocks always produce a carry bit. Propagate blocks produce a carry if and only if they get a carry in. Blocks can do neither - will always produce a 0 carry.

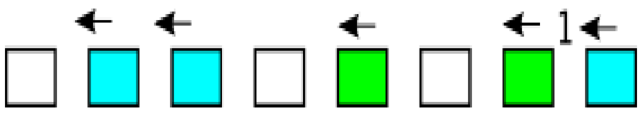
First, we add together all the digits individually without any carry to work out the generate and propagate behaviors for each individual place in the sum. If those places would generate a carry, we mark them with a "generate". If they don't carry over, but adding on another 1 would make them create a carry, we mark them with "propagate". A bit marked as propagate will create a carry bit out only if it gets a carry bit in.

0 + 0 = 0, no carry. 0 + 1 doesn't carry either - doesn't generate or propagate.

0 + 1 = 1, no carry. 1 + 1 would create a carry - doesn't generate but does propagate.

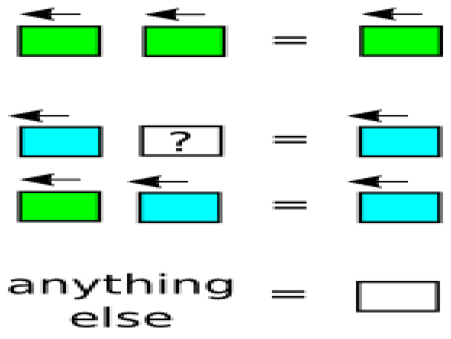
1 + 0 = 1, same as last line

1 + 1 = 0, carry 1. 0 + 1 doesn't carry - generates but doesn't propagate.

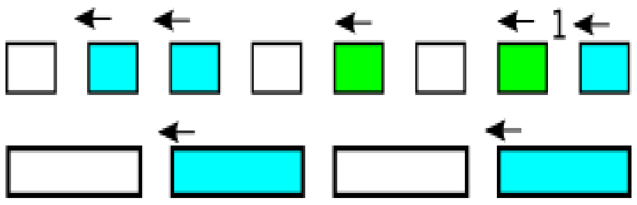


Each place in the sum becomes a stretch 1 place long.

Two adjacent stretches can be combined together in to a larger range, which will propagate if both blocks within it propagate. It will generate if either the higher place block generates, or the lower place block generates and the larger place block propagates the carry.

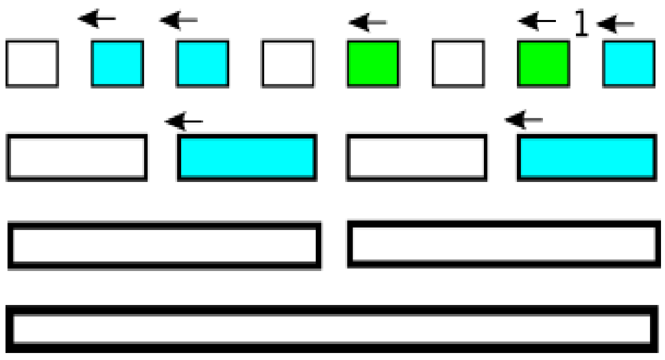


We can combine two stretches next to each other to form a larger range. We combine together pairs of places using those rules to find behavior of larger stretches.



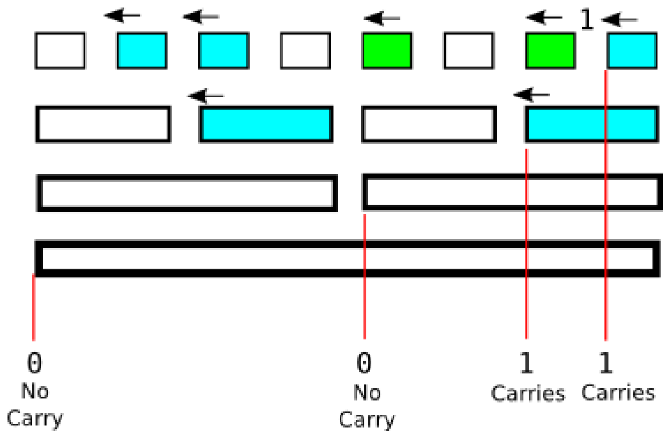
Combined ranges - each 2 places long.

By combining stretches together, we get the behaviour of large blocks. In the first layer, we could combine together pairs of places, in the second layer we combine together the pairs to form ranges of 4, and could continue moving up to 8, 16 and so on. The gate delay through this is the depth of the tree, which now only increases by a level every time the number of bits in the sum doubles. With ripple carry, the amount of time would double if the number of bits doubled.



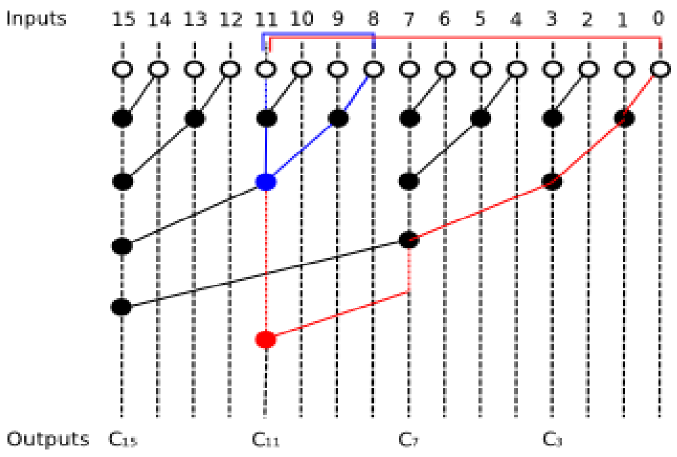
We can combine blocks to get the behavior of the whole range.

So what use is this? We only have these "propagate" and "generate" values for the ranges here. The key is that if we have a range from the smallest bit up to any point, the carry out of the top bit is 1 if and only if the range generates. In our example above, we have four ranges which extend all the way to the least significant bit on the right, giving us the carries out of the 1st, 2nd, 4th and 8th bit from the right.



For a range from the least significant bit up to any other bit, the generate flag tells you whether there is a carry.

That completes the theory behind how tree adders fundamentally work. In practice, we often want to have carries out of places other than powers of 2 so build the tree so that the ranges overlap at each level. There are many different designs of how to arrange the tree to create carries at different points. In particular, if we get the carry out of every place then we can recombine them with the original "propagate" values for the next place up to get the numbers added. The way these different designs tend to be drawn is as a tree, where we mark each range at the point where it ends. Each range is built from the previous range to end at that position, and another adjacent range.



Standard tree notation, with two of the ranges highlighted. The red and blue trees have the links you would follow to find where they start and end highlighted in their color. The red gets all the way to 0- gives us the carry out of its column.

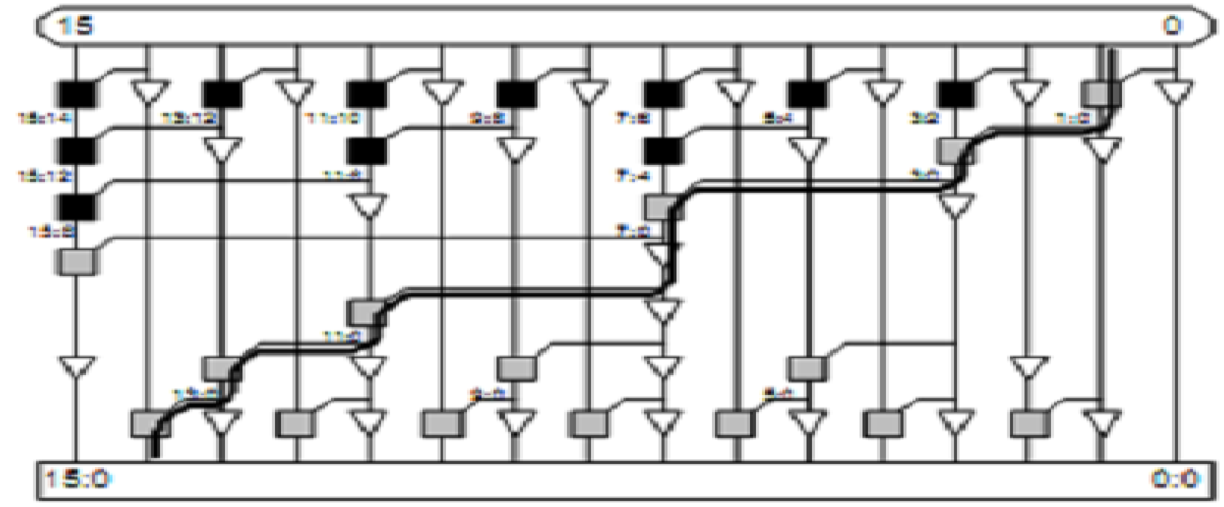
Notice that we can still only combine ranges which are next to each other - the red range combines the blue range and another which begins the place after the blue one ends. For details of specific designs I'd recommend Hardware algorithms for arithmetic modules. This uses the tree notation described above.

This page generates adders based on the Kogge-Stone tree design, but rather than generating carries from every bit (as the original does), this trims it down to provide only a few at regular intervals. This attempts to allow you to strike a balance between fast but complicated trees, which can have lots of connections crossing over each other and be difficult to lay out in hardware, and slower but simpler ripple-carry adders, which can actually be faster in FPGA designs.

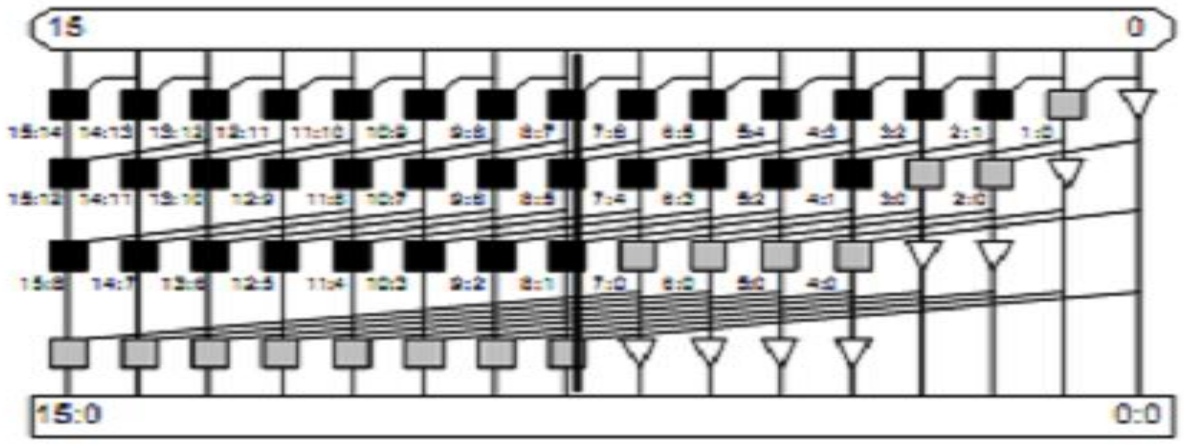
III. CARRY TREE ADDERS

The various types of carry tree adders are shown in Fig.2. Each carry tree adder consists of three parts. They are: Upper part, Middle part, Lower part. Using these parts the carry tree adders computes N outputs from N inputs as shown in Fig.1. The Upper part generates and propagates the carry signal from the input to the prefix stage using the formula given in equation (4). The propagated and generated carry signals are combined using the associate operator “o”. This operation is performed in the middle part using the formula given in equation (5). The Middle part consists of prefix cells such as black cells, grey cells and white buffers [1]. The arrangement of these prefix cells in different order results in various types of Carry Tree adders. Where the carry signals need not to be propagated. Such operations are performed by grey cells. The grey cells generate the carry signal only. Black cell generates and propagates the carry signal. There are some places the white buffers are used to reduce the loading effect for the further stages.

The Lower part generates the overall sum using the formula given in equation (6). Depends on the arrangement of prefix cells, the carry tree adders involves in tradeoffs like area, power, delay, interconnect count, fan-out and logic depth [3 & 4]. Fig.2 (a) shows the Brent-Kung the dark black line in the figure indicates the critical path of the adder. The critical path for Han-Carlson and Kogge-Stone are less. So these two adders are expected to be the fastest adder. The power utilized by all the Carry Tree Adder is more than the Simple Adder.

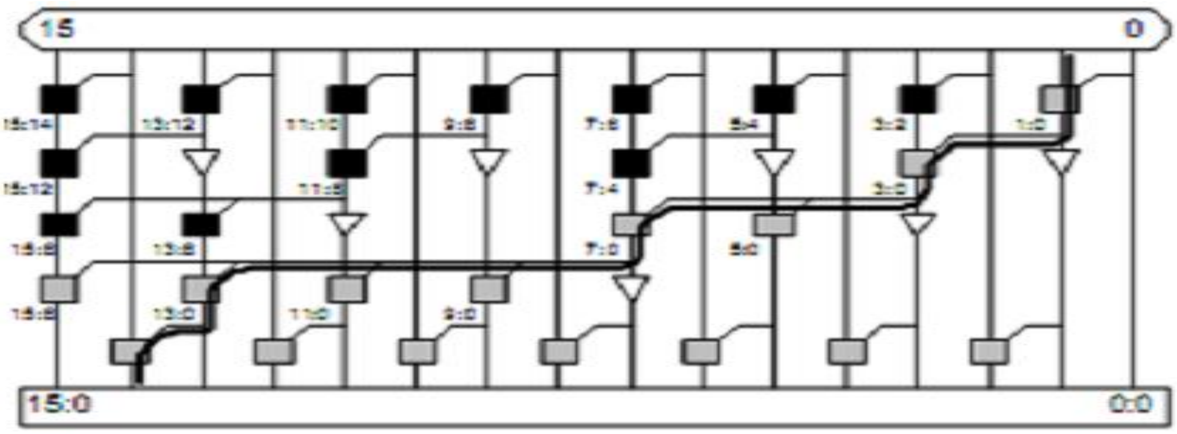


(a)Brent Kung

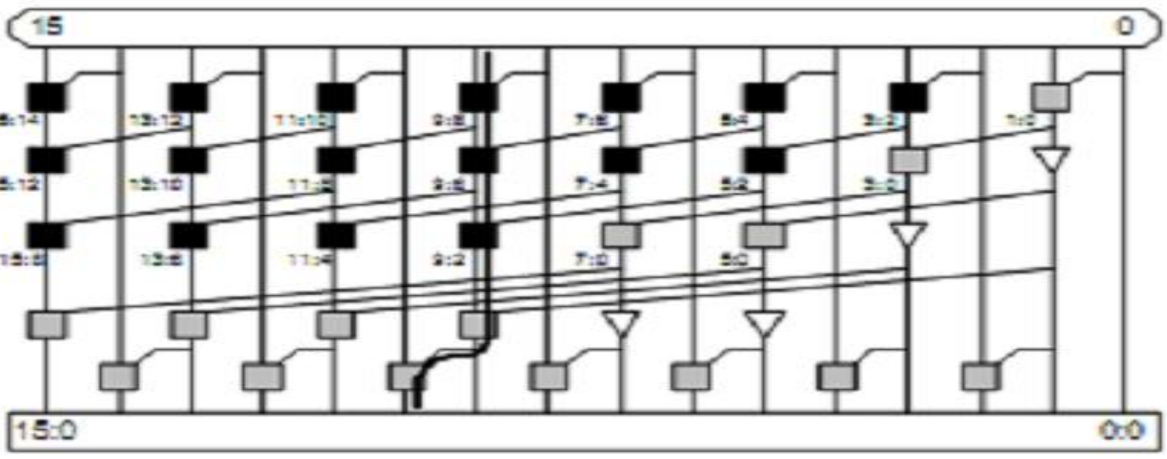


(b) Kogge Stone

minimum area and maximum logic depth. Due to the maximum logic depth, the delay of this adder is expected to be high. Fig.2 (b) shows the Kogge-Stone adder. Adder. It is designed in such a way that it provides it provides maximum interconnect count and area but minimum logic depth and fan-out. Ladner-Fischer adder as shown in Fig.2 (c) provides minimum logic depth with improved area. Han-Carlson adder as shown in Fig. 2(d) provides minimum logic depth and minimum interconnect count.



(c) Lander Fischer



(d)Han Carlson

Fig.2. (a-f) Carry Tree adders

Simple adder is designed using Verilog HDL '+' operator. The carry chain structure on FPGA makes Simple Adder to provide high performance. But this is not an efficient adder for VLSI implementation. In this paper, Carry Tree Adder is compared with Simple Adder for both ASIC and FPGA implementation. Parallel-prefix adders, also known as carry tree adders and pre compute the status and generate signals. These signals are variously combined using the fundamental carry operator (fco).

$$(g_L, p_L) \circ (g_R, p_R) = (g_L + p_L{\bullet}g_R,\ p_L \bullet p_R) \quad \textbf{(4)}$$

Due to associative property of the fundamental carry operator, these operators can be combined and different ways to perform various adder structures. For, example the four-bit carry-look ahead generator is given by:

$$c_4 = (g_4, p_4) \circ [\ (g_3, p_3) \circ [(g_2, p_2) \circ (g_1, p_1)]\ ] \quad \textbf{(5)}$$

A simple rearrangement of the order of operations allows equal operation; perform in a more efficient tree structure for this four bit example:

$$c_4 = [(g_4, p_4) \circ (g_3, p_3)] \circ [(g_2, p_2\ ) \circ (g_1, p_1)] \quad \textbf{(6)}$$

It is readily apparent that a key advantage of the tree-structured adder is that the critical path due to the carry delay is on the order of log2N for an N-bit wide adder. Then the arrangement of the prefix work connection gives rise to various families of adders. Then the discussion of the various carry tree structures,

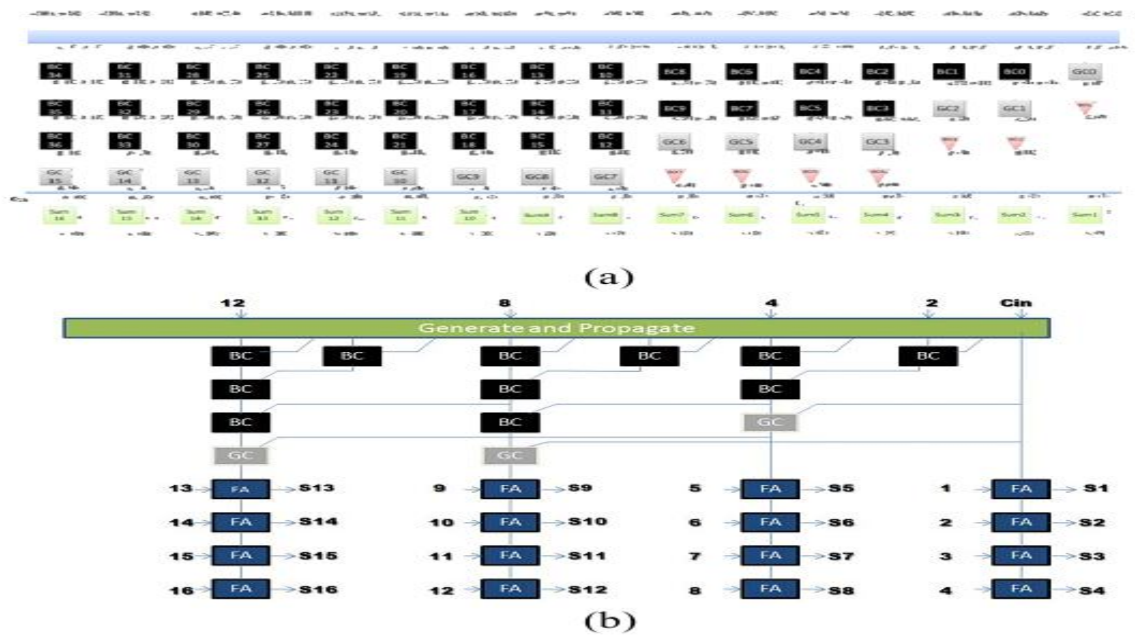


Fig.3. (a) 16 bit Kogge-Stone adder and (b) sparse 16-bit Kogge-Stone adder.

For this study, the focus is on the Kogge-Stone adder, known for having minimal logic depth and fan out (see Fig 3(a)). Here we designate BC as the black cell which generates the ordered pair in equation (5); the gray cell (GC) generates the left signal only, following .The interconnect area is known to be high, but for an FPGA with large routing overhead to begin with, this is not as important as in a VLSI implementation. The regularity of the Kogge-Stone prefix network has built in redundancy which has implications for fault-tolerant designs .The sparse Kogge-Stone adder, shown in Fig 1(b), is also studied. This hybrid design completes the summation process with a 4 bit RCA allowing the carry prefix network to be simplified.

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 change with 4-bit RCA. it is interested to compare with the performance of this adder with the sparse and regular Kogge-Stone adders. These also of interest for the spanning tree CLA is its testability features.

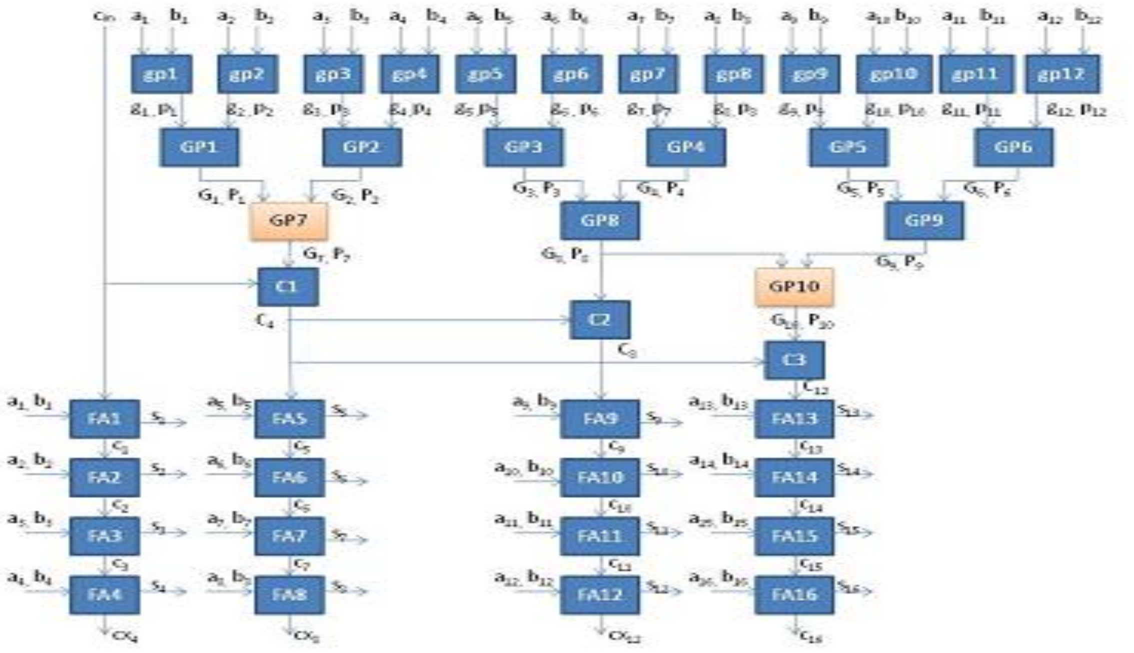


Fig. 4. Spanning Tree Carry Look ahead Adder (16 bit)

IV.RESEARCH AND PROPOSED WORK

The different types of carry tree adders are discussed in [4]. In [5], the authors implemented different types of adders like Simple Adder, Carry Look Ahead Adder, Carry Skip Adder, and Carry Select Adder on the Virtex2 FPGAs and found that the Simple Adder provides better performance. In [3], the authors discussed various parallel prefix networks design and implementation on a Xilinx Virtex5 FPGA. It is observed that the Simple Adder provides better performance over the prefix networks for the bit widths up to 256 bits. This is due to the advantage of the carry chain structure on the FPGA. All these works by different authors shows that the simple adder provides better performance on FPGA.

The area, delay results for these works depend upon synthesis reports. In [2], the authors described several Carry Tree Adders implemented on a Xilinx Spartan3 EFPGA. It is found that the Kogge Stone Carry Tree Adder provide better delay performance for the higher order bits. The results obtained for this paper is similar to those presented in [2]. Carry Tree Adders are designed, coded, simulated and synthesized and then it is compared with the Simple Adder. The obtained area, power, delay results of various Carry Tree Adders are compared with each other and also with the Simple Adder. Among all the Carry Tree Adders, Kogge-Stone Adder and Han-Carlson Adder is expected to be the fastest adder in ASIC implementation but not in FPGA implementation.

In this paper, Kogge-Stone Adder is taken, since it is having minimum fan-out and logic depth than Han-Carlson Adder, and modified using Fast Carry Logic technique in order to make it suitable for FPGA implementation [6, 7, 8 &9]. The addition operation performed by Simple Adder, which is generated by synthesis tool, is shown in Fig.5 (a). From Fig.5 (a), it is clear that the Prefix-computation stage of the Simple Adder uses multiplexers. Similarly, the Prefix-computation stage of Carry Tree Adder is replaced with the Fast Carry logic technique which uses muxes as shown in Fig.5 (b). The Fast Carry Logic architecture for 4-bit addition is shown in Fig.5(c). Instead of using Black cells, Grey cells and White buffers to propagate and generate the carry signals, simple muxes are used. The blocks present in Fast Carry Logic technique also uses muxes. The input to the Fast Carry Logic is the propagated and generated carry signal of the Pre-computation stage. The Pre-computation and Post-computation of the modified adder is similar to that of the normal carry tree adders.

V. RESULTS

The delay, power and cell area results obtained by synthesizing the designed adders for 128bits using Cadence RTL compiler (90nm technology) is shown in Table 1, 2 & 3. The abbreviations used in the table are: KS for the Kogge Stone Adder, BK for the Brent Kung Adder, LF for the Lander Fischer Adder and HC for Han Carlson Adder.

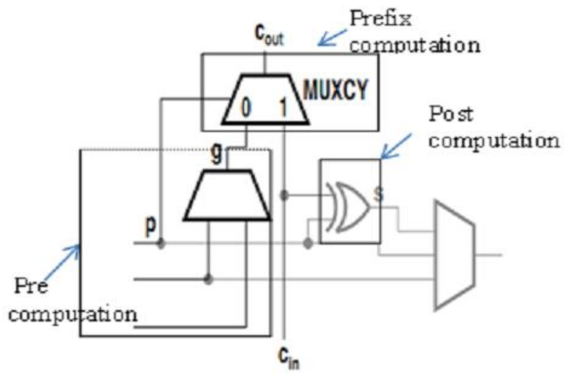


Fig.5 (a) Simple Adder

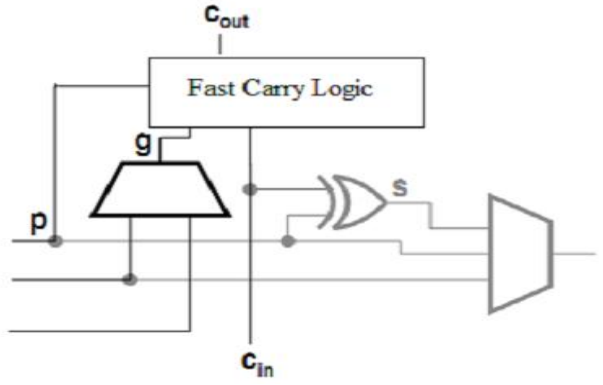


Fig.5 (b) Carry Tree Adder

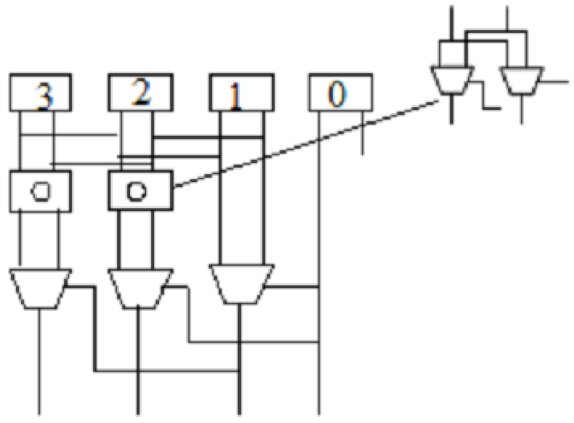


Fig.5(c) Fast Carry Logic for 4-bit Carry tree addition

The delay is measured in terms of nanoseconds, power in terms of nanowatt. From the results it is found that the Carry Tree Adders provide best delay performance than the Simple adder. Among the Carry Tree Adders, Kogge-Stone Adder and Han-Carlson Adder provide best delay as it is expected but the area and power utilized by those adders are more. Comparatively, Brent-Kung Adder and Lander-Fischer Adder utilizes less area and power.

Table 1: Delay Results of Carry Tree Adders compared with Simple Adder

N	SIMPLE ADDER	CARRY TREE ADDERS			
		KS	BK	LF	HC
8	796	507	790	568	567
16	1685	599	762	684	670
32	3420	817	980	830	871
64	6854	886	1092	1090	978
128	8900	915	1868	1572	1079

Table2: Power Results of Carry Tree Adders compared with Simple Adder

N	SIMPLE ADDER	CARRY TREE ADDERS			
		KS	BK	LF	HC
8	9646.743	20638.01	15276.11	15284.52	15958.7
16	21865.57	46160.68	32465.95	33352.82	36415.43
32	45429.39	114339.5	70614.87	74515.97	83159.26
64	90625.67	242942.6	138459.4	158185.3	174922.5
128	252180.9	563905.3	381100.1	396825.7	294698.2

Table3: Cell Area Results of Carry Tree Adders compared with Simple Adder

N	SIMPLE ADDER	CARRY TREE ADDERS			
		KS	BK	LF	HC
8	119	291	215	215	223
16	254	675	454	463	505
32	509	1573	943	1002	1137
64	1029	3672	1911	2224	2537
128	2903	8385	3906	4948	5608

Fig.4 shows the simulated delay results of the adders for the bit widths up to 128bits using Xilinx ISE13.2 software tool. From the Fig.4,it is found that the Simple Adder provide best delay performance over the Carry Tree Adder. The obtained delay result is entirely different from the result shown in Table 1. This is because of the presence of Fast Carry chain structure on Xilinx FPGA. Among the Carry Tree adders, Kogge-Stone Adder provides best delay as it is expected.

Fig.5 (a-d) shows the delay results of Kogge-Stone Adder, Kogge-Stone Modified Adder and Simple Adder for the FPGA families like Spartan-3E, Virtex-4, Virtex-5 and Virtex-6 Lower power. Some of the 64-bit adder structure cannot be fitted in to all the devices under this family.

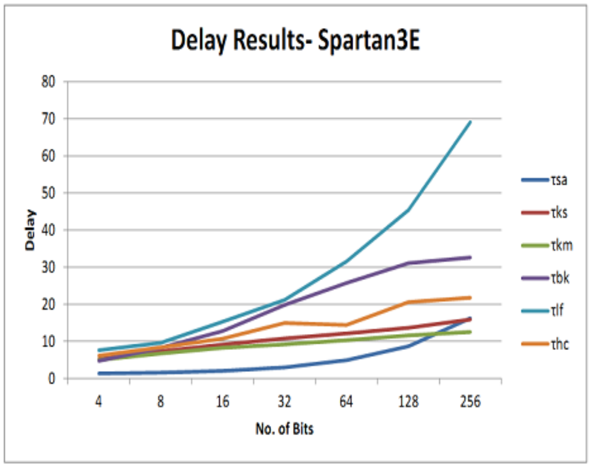


Fig.6. Simulated Delay Results of Carry Tree Adders compared with Simple Adder

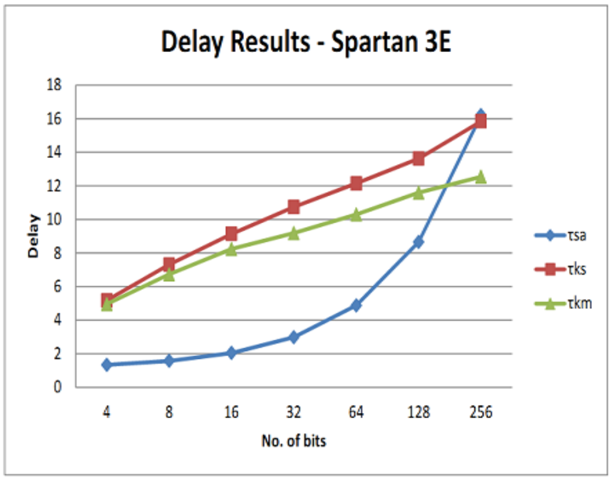


Fig.7.a)

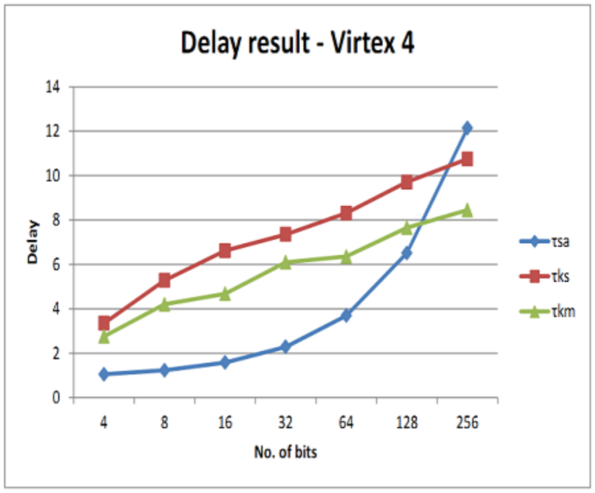


Fig.7.b)

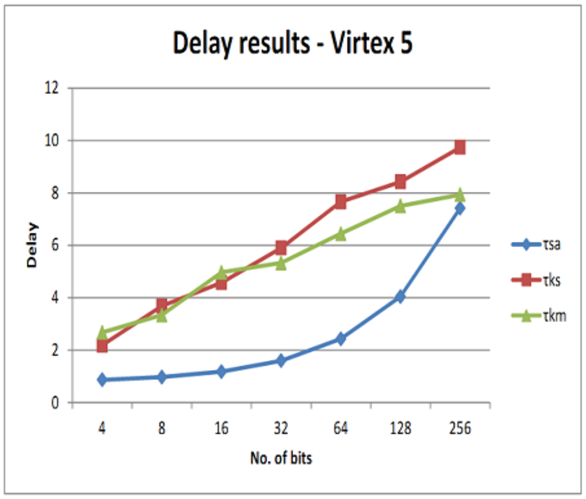


Fig.7.c

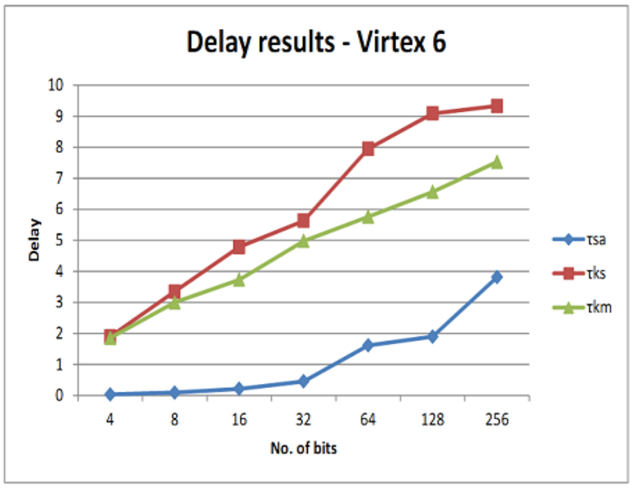


Fig.7.d

Fig.7. (a-d) Simulated Delay results.

Depends on the adder structure, the device and package has been selected. From the Fig it is found that, for Spartan-3E FPGA, Kogge-Stone adder provide best performance after it reaches 256 bits whereas Modified adder provides best performance after it reaches 128bits, for Virtex-4 FPGA, Kogge-Stone adder provides best performance after it reaches 128bits whereas Modified adder provides best performance from 128bits, for Virtex-5 FPGA, Kogge-Stone adder provides best performance after it reaches 256bits whereas Modified adder provides best performance from 128bits, for Virtex-6 FPGA, it is able to reduce the delay of Carry Tree Adder but Simple Adder provide better delay performance.

VI. METHOD OF STUDY

The adders to be studied were designed with varied bit widths up to 128 bits and coded in vhdl. The functionality of the designs is verified via simulation with the Model Sim. The Xilinx ISE 12.2 software was used to synthesize the designs onto the Spartan3E FPGA. Then in order to effectively test for the critical late process, two steps were taken. At First block (labeled as ROM in the figure below) was instantiated on the FPGA using the Core Generator to allow arbitrary patterns of inputs to be applied to the adder design. A multiplexer design at each adder o/p selects where or not to include the adder in the measured results can be shown in Fig.3. Switch on the FPGA board was wired to the select pin of the multiplexers. This allows the measurements to made the subtract out the delay due to the memory, the multiplexers, and interconnect.

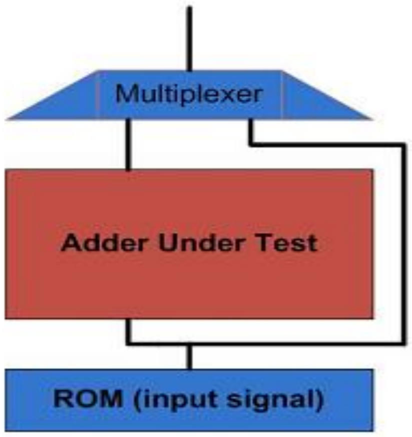


Fig.8. Circuit used to test the adders.

Second, the parallel prefix network was analyzed to determine if a specific pattern could be used to extract the worst case delay. Considering the structure of the Generate-Propagate (GP) blocks (i.e., the BC and GC cells), we were able to develop the scheme by considering the subset of input values to the GP blocks is as fallows.

Table 4: Subset of (g, p) Relations Used for Testing

(g_L, p_L)	(g_R, p_R)	$(g_L + p_L\, g_R, p_L\, p_R)$
(0,1)	(0,1)	(0,1)
(0,1)	(1,0)	(1,0)
(1,0)	(0,1)	(1,0)
(1,0)	(1,0)	(1,0)

If we arbitrarily assign the (g, p) ordered pairs the values (1, 0) = True and (0, 1) = False, then the table is self-contained and forms an OR truth table. If the both inputs to the GP block is False, then the output is False conversely. if both inputs are True and output is also True. Hence the input patterns that alternates between the (g, p) pairs of (1, 0) and (0, 1) will force its GP pair block to the alternate states. Like as it is easy to see that the GP blocks being fed by its predecessors will also alternate states. This scheme will ensure that the worsted case delay will be generated in the parallel prefix network since every block will be active. In order to ensure this scheme works, the parallel prefix adders were synthesized with the “Keep Hierarchy” design setting turned on (otherwise, the FPGA compiler attempts to reorganize the logic assigned to each LUT). With this option turned on, it ensures that each GP block is mapped to one LUT, preserving the basic parallel prefix structure, and ensuring that this test strategy is effective for determining the critical delay. The designs were also synthesized for speed rather than area optimization.

The adders were tested with a Tektronix TLA7012 Logic Analyzer. The logic analyzer is equipped with the 7BB4 module that provides a timing resolution of 20 ps under the Magni Vu setting. This allows direct measurement of the adder delays. The Spartan 3E development board is equipped with a soft touch-landing pad which allows low capacitance connection directly to the logic analyzer. The test setup is depicted in the figure below.

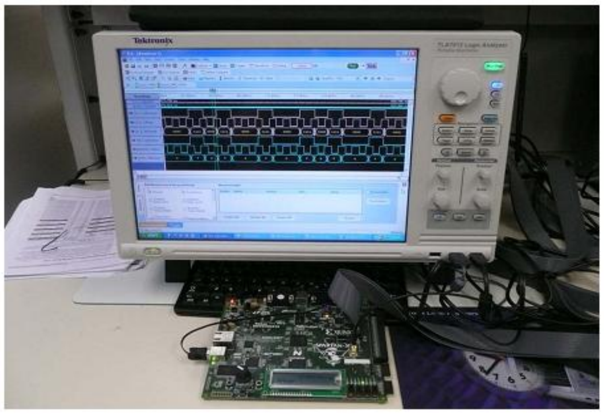


Fig.9. Test setup showing the Logic Analyzer and Spartan 3E development board.

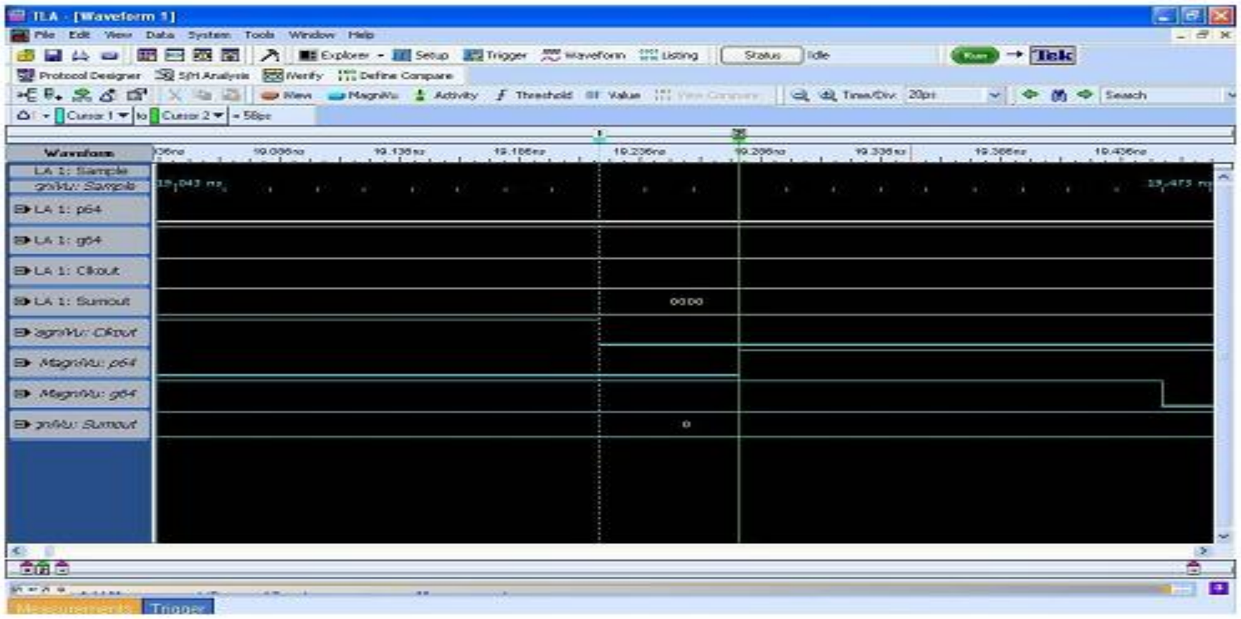


Fig.10. Screen shot of a delay measurement for a 64 bit adder using MagniVu timing (blue traces) on the TLA 7012.

V. DISCUSSION OF RESULTS

The simulated adder delays obtained from the Xilinx ISE synthesis reports are shown in Fig. 11. The simulation results for the carry skip adders are not included because the ISE software is not able to correctly identify the critical path through the adder and hence does not report accurate estimates of the adder delay. Observe that a semi-log plot is employed, so as expected the tree-adder delay plots as a straight line on this graph. Somewhat surprising is the fact that the sparse Kogge-Stone adder has about the same delay as the regular Kogge-Stone adder. Because the sparse Kogge Stone completes the summation process with a 4 bit RCA, which are optimized via the fast carry chain, its performance is expected to be intermediate between the regular Kogge-Stone adder and the RCA. The impact of the routing overhead would seem to be a likely cause.

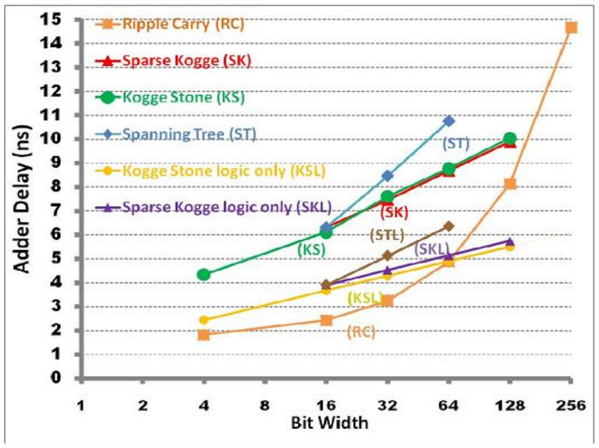


Fig. 11. Simulation results for the adder designs.

However, according to the synthesis reports, the delay with the logic only makes the regular Kogge-Stone slightly faster. This will need to be a topic of further investigation. Overall, when the delay due to routing overhead is deleted, then the tree adders are closer to the simple RCA design. Then the RCA adders exhibits the delay with widths up to 64 bits when the routing delay is excluded and out to 128 bits with the routing delay included.

Figures 12 and 13 depict the measured results using TLA. The comparison between the tree adders ,RCA is given in Figure12. Then the basic trends for the same: the tree adders exhibit logarithmic delay dependence on bit widths and the RCA has linear performance. An RCA as large as 160 bits wide was synthesizable on the FPGA, while a Kogge-Stone adder up to 128 bits wide was implemented. The carry-skip adders are compared with the Kogge-Stone adders and the RCA in Figure 13. Carry skip adders with a skip of four and eight were implemented. The poor performance of the carry skip adders is attributable to the significant routing overhead incurred by this structure.

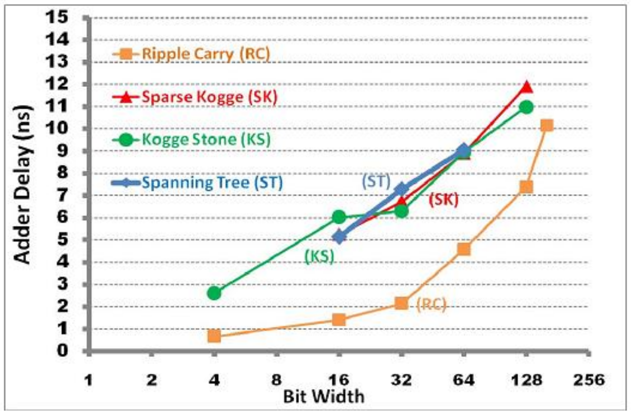


Fig.12. Measured results for the parallel-prefix adder designs compared with the RCA.

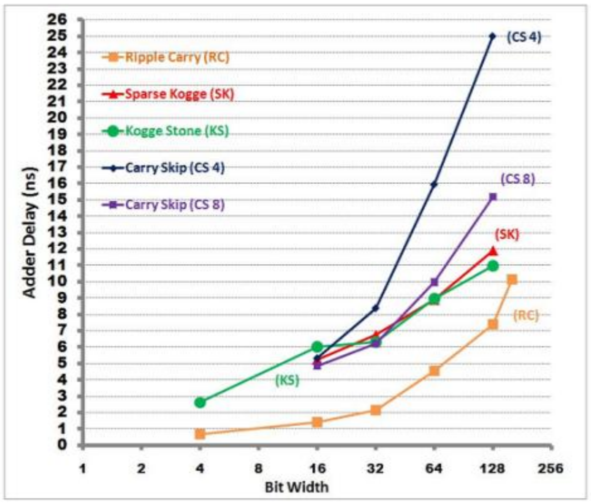


Fig.13. Measured results for the carry-skip adders compared to the RCA and Kogge-Stone adders.

The actual measured data appears to be a bit smaller than what is predicted by the Xilinx ISE synthesis reports. An analysis of these reports, which give a breakdown of delay due to logic and routing, would seem to indicate that at adder widths approaching 256 bits and beyond, the Kogge-Stone adder will have superior performance compared to the RCA. Based on the synthesis reports, the delay of the Kogge-Stone adder can be predicted by the following equation:

$$t_{KS} = (n+2)\cdot\Delta_{\mathrm{LUT}} + \rho_{\mathrm{KS}}(n) \qquad \textbf{(7)}$$

where N= 2n, the adder bit width, Δ_{LUT} is the delay through a lookup table (LUT), and $_{\rho KS}$(n) is the routing delay of the Kogge-Stone adder as a function of n. The delay of the RCA can be predicted as:

$$t_{RCA} = (N-2)\cdot\Delta_{\mathrm{MUX}} + \tau_{\mathrm{RCA}} \qquad \textbf{(8)}$$

where Δ_{MUX} is the mux delay associated with the fast-carry chain and τ_{RCA} is a fixed logical delay process. There is no routing delay assumed for the RCA due to the use of the fast carry chain ,For the Spartan3E FPGA, the synthesis of reports give the following values: Δ_{LUT}= 0.612, Δ_{MUX}= 0.051, and τ_{RCA}= 1.715. Even though the $\Delta_{MUX} << \Delta_{LUT}$, it is already expected that the Kogge Stone adder will eventually be faster than the RCA because N= 2N provided that $_{\rho KS}$(n) grows , it can be relatively slower than (N – 2) Δ_{MUX}. Indeed, Table II predicts that the Kogge Stone adder will have superior performance at N = 256.

Table II
Delay Results for the Kogge-Stone Adders

N	Synth. Predict	Route Delay	Route Fitted	Delay t_{KS}	Delay t_{RCA}
4	4.343	1.895	1.852	4.300	1.817
16	6.113	2.441	2.614	6.286	2.429
32	7.607	3.323	3.154	7.438	3.245
64	8.771	3.875	3.800	8.696	4.877
128	10.038	4.530	4.552	10.060	8.141
256	–	–	5.410	11.530	14.669

(all delays given in ns)

The second and third columns represent the total predicted delay and the delay due to routing only for the Kogge Stone adder from the synthesis reports of the Xilinx ISE. The fitted routing in columns represents the predicted routing delay using a quadratic polynomial in n based on the n= 4 to 128. This allows the n= 256 routing late to be predicted with some degree of confidence as an actual Kogge-Stone adder at this bit width was not synthesize. Then the final two columns give the predicted adder delays for the Kogge-Stone and RCA using equations (7) and (8), respectively. The good match between the measured and simulated data for the implemented Kogge-Stone adders and RCAs gives confidence that the predicted superiority of the Kogge-Stone adder at the 256 bit width is accurate.

This differs from the results ,where the parallel-prefix adders, including the Kogge-Stone adder, always exhibited inferior performance compared with the RCA (simulation results out to 256 bits were reported). The work ,did use a different FPGA (Xilinx Virtex 5), which may account for some of the differences.

VII. CONCLUSION

This paper presents a new approach for the basic operators of parallel prefix tree adders. In Skalansky, KS, LF, Knowles adders the delay is reduced by this new approach, in BK adder there is no much difference with this new approach and in the case of HC adder the delay is increased. The same can be understood with reference to number of logic levels of implementation, as the logic levels are more delay increases. The area requirement can be considered from the utilization of LUTs, Slices and over all gate count. The BK adder occupies less area compared to other adders, but does not show much difference with new approach. Skalansky, LF adders occupies slightly more area in new approach compared to old method. KS and Knowles adders occupy more area in new approach. HC adder shows almost no difference with new approach.

VIII.REFERENCES

[1] N. H. E. Weste and D. Harris, CMOS VLSI Design, 4th edition, Pearson–Addison-Wesley, 2011.

[2] R. P. Brent and H. T. Kung, “A regular layout for parallel adders,” IEEE Trans. Comput., vol. C-31, pp. 260-264, 1982.

[3] D. Harris, “A Taxonomy of Parallel Prefix Networks,” in Proc. 37th Asilomar Conf. Signals Systems and Computers, pp. 2213–7, 2003.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] M. Bečvář and P. Štukjunger, “Fixed-Point Arithmetic in FPGA,” Acta Polytechnica, vol. 45, no. 2, pp. 67- 72, 2005.

[10] 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.

E. Sreenivasa Goud was born at Chinnatekur, Kurnool, A.P, India, on 1st October 1985. He completed Diploma in Electronics & Communication Engineering in 2005 from Govt. Polytechnic College, Anantapur, A.P, India. He received B.Tech. degree in Electronics and Communication Engineering from SKTRM College of Engineering, Kondair, Mahaboob Nagar, A.P, India, in 2010. He is pursuing his M.Tech. in VLSI, from Kottam College of Engineering. Kurnool, A.P, India. His area of interest includes VLSI design and Digital Communication.

P.C.Praveen Kumar was born at Kurnool, A.P. He received Bachelor degree in Electronics and Communication Engineering from SKTRM College of Engineering, in 2007. He received his Master degree in Digital Electronics & Communication Systems in 2010 from JNTUH. Hyderabad, A.P. India. He is presently pursuing his Ph.D. work on Inverted L Antennas from Gitam University, Vishakhapatnam, A.P, India. Presently he is working as Head of the department of electronics and communication engineering in Kottam Group of Institutions, Kurnool, A.P, India. He is having 5 years of teaching experience. His area of interest includes VLSI design, low power testing, Electromagnetic Fields and Transmission Lines, Antenna Theory and Wave Propagation & Microwave Technology.