# An Improved Optimization Techniques for Parallel Prefix Adder using FPGA

# O.Anjaneyulu<sup>1</sup>, L.Swapna<sup>2</sup>, C.V.Krishna Reddy<sup>3</sup>,

Member, IEEE KITS, Warangal<sup>12</sup>, NNRESGI-Hyd<sup>3</sup> Andhra Pradesh, India

**ABSTRACT**—In this paper, Carry Tree Adders are Proposed. Parallel prefix adders have the best performance in VLSI Design. Parallel prefix adders gives the best performance compared to the Ripple Carry Adder (RCA) and Carry Skip Adder (CSA). Here Delay measurements are done for Kogge-Stone Adder, Sparse Kogge-Stone Adder and Spanning Tree Adder. Speed of Kogge-Stone adder and Sparse Kogge-Stone adder have improved compared to the Ripple Carry Adder (RCA) and Carry Skip Adder (RCA) and Carry Skip Adder (CSA). Model Simulator-Altera 6.6d and Xilinx 10.1 tools were used for simulation and synthesis of the design.

*Index Terms* –*Carry Skip Adder (CSA), Kogge-Stone adder, Ripple carry adder (RCA), sparse Kogge-Stone adder and Spanning tree adder.* 

## I. INTRODUCTION

In 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. 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).

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). Finally, some conclusions and suggestions for improving FPGA designs to enable better tree-based adder performance are given.

#### II. CARRY-TREE ADDER DESIGNS

Parallel-prefix adders, also known as carry-tree adders, pre-compute the propagate and generate signals [1]. These signals are variously combined using the *fundamental carry operator* (fco) [2].  $(G_L, P_L) \square (G_R, P_R) = (G_L + P_L \bullet G_R, P_L \bullet P_R)(1)$ 

Due to associative property of the fco, these operators can be combined in different ways to form various adder structures. For, example the four-bit carry-look ahead-generator is given by:  $c_4=(g_4, p_4) \square [(g_2, p_3) \square [(g_2, p_2) \square (g_1, p_1)]]$  (2)

A simple rearrangement of the order of operations allows parallel operation, resulting in a more efficient tree structure for this four bit example:

 $c_4 = [(g_4, p_4) \square (g_3, p_3)] \square (g_2, p_2) \square (g_1, p_1)]$ (3)

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. The arrangement of the prefix network gives rise to various families of adders. For a discussion of the various carry-tree structures, see [1,3].

For this study, the focus is on the Kogge-Stone adder [4]

Here we designate BC as the black cell which generates the ordered pair in equation (1); the grey cell (GC) generates the left signal only, following [1]. 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 [5]. The sparse Kogge-Stone adder, shown in Fig 2, 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 [6]. 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. Also of interest for the spanning-tree CLA is its testability feature [7].



Fig3. Spanning Tree Carry Look ahead Adder (16 bit)

www.ijmer.com Vol. 3, Issue. 5, Sep - Oct. 2013 pp-3107-3115

ISSN: 2249-6645

# III. 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 were verified via simulation with Model Simulator. The Xilinx ISE 10.1 software was used to synthesize the designs onto the Spartan 3E FPGA. In order to effectively test for the critical delay, two steps were taken. First, a memory block (labelled 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 at each adder output selects whether or not to include the adder in the measured results, as shown in Fig A switch on the FPGA board was wired to the select pin of the multiplexers. This allows measurements to be made to subtract out the delay due to the memory, the multiplexer. And interconnect (both external cabling and internal routing).

Second, the parallel prefix network was analysed 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 following scheme, by considering the following subset of input values to the GP blocks.

(gL,pL) (gR,pR)	(gL + pL gR, pL pR)
(0,1) (0,1)	(0,1)
(0,1) (1,0)	(1,0)
(1,0) (0,1)	(1,0)
(1,0) (1,0)	(1,0)

**Table1:** Subset of (g, p) Relations Used for Testing

If we arbitrarily assign the (g, p) ordered pairs the values (1,0) = True and (0, 1) = False, then the table is selfcontained and forms an OR truth table. Furthermore, if both inputs to the GP block are False, then the output is False; conversely, if both inputs are True, then the output is True. Hence, an input pattern that alternates between generating the (g, p) pairs of (1, 0) and (0, 1) will force its GP pair block to alternate states. Likewise, it is easily seen that the GP blocks being fed by its predecessors will also alternate states. Therefore, this scheme will ensure that a worse 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.

#### IV. DISCUSSION OF RESULTS

The simulated adder delays obtained from the Xilinx ISE synthesis reports are shown in Fig. 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. 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:

 $t\kappa s = (n+2) \square \square u u + \square \square (n)$  (4) where N = 2n, the adder bit width,  $\Delta LUT$  is the delay through a lookup table (LUT), and  $\rho \kappa s(n)$  is the routing delay of the kogge-Stone adder as a function of *n*. The delay of the RCA can be predicted as:  $tR cA = (N-2) \square \square u x + \square R cA$  (5)

where  $\Delta MUX$  is the mux delay associated with the fast-carry chain and  $\tau R c A$  is a fixed logic delay. There is no routing delay assumed for the RCA due to the use of the fast-carry

chain. For the Spartan 3E FPGA, the synthesis reports give the following values:  $\Delta LUT = 0.612$  ns,  $\Delta MUX = 0.051$  ns, and T R C A = 1.715 ns. Even though  $\Delta MUX \ll \Delta LUT$ , it is expected that the Kogge-Stone adder will eventually be faster than the RCA because N = 2n, provided that  $\rho \kappa s(n)$  grows relatively slower than  $(N - 2) \Box \Delta MUX$ . Indeed, Table II predicts that the Kogge-Stone adder will have superior performance at N = 256.

4 16 32	Predict 4.343	Delay 1.895	Fitted	tKS	tRCA
4 4 16 9	4.343	1.895	1.050		
16 32			1.852	4.300	1.817
32	6.113	2.441	2.614	6.286	2.429
54	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

Table2 : Delay	<b>Results for the Kogg</b>	e-Stone Adders

(all delays given in ns)

#### www.ijmer.com Vol. 3, Issue. 5, Sep - Oct. 2013 pp-3107-3115 ISSN: 2249-6645 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 software. The fitted routing delay in column four represents the predicted routing delay using a quadratic polynomial in n based on the N = 4 to 128 data. This allows the N = 256 routing delay to be predicted with some degree of confidence as an actual Kogge-Stone adder at this bit width was not synthesized. The final two columns give the predicted adder delays for the Kogge-Stone and RCA using equations (4) and (5), 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 in [10], 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 in [10] did use a different FPGA (Xilinx Vertex 5), which may account for some of the differences. The poor performance of some of the other implemented adders also deserves some comment. The spanning tree adder is comparable in performance to the Kogge-Stone adder at 16 bits. However, the spanning tree adder is significantly slower at higher bit widths, according to the simulation results, and slightly slower, according to the measured data. The structure of the spanning tree adder results in an extra stage of logic for some adder outputs compared to the Kogge-Stone. This fact coupled with the way the FPGA place and route software arranges the adder is likely the reason for this significant increase in delay for higher order bit widths. Similarly, the inferior performance of the carry-skip adders is due to the LUT delay and routing overhead associated with each carry-skip logic structure. Even if the carry-skip logic could be implemented with the fast-carry chain, this would just make it equivalent in speed to the RCA. Hence, the RCA delay represents the theoretical lower limit for a carry-skip architecture on an FPGA.

Messages											
₽-∲ (Ripple_Carry/A	0010110111010101	0000111111	10000	0000110101	10011			0010110111	010101		
🛃 / Ripple_Carry/B		0001111000	11110	0010101101	)11110			0010110010	00110	0010110011	011110
👌 /Ripple_Carry/Cin	30										
🛃 / Ripple_Carry/SUM		0010111000	101110	0011100011	)1001	0011100011	010010	0101101001	11100	0101101010	10011
🕴 /Ripple_Carry/Cout	90										
🚽 /Ripple_Carry/c		0001111111	1000	0000111101	11110	0000111101	11111	0010110110	00111	0010110111	011100

V. SIMULATION RESULTS

Nessages											
₽-∲ (carry_select_adder/a	0010111100111100	0000110011	001100			0000001100	11100	0010111100	11100		
₽� (cany_select_adder/b	0011001111001111	000001100	10011			0011111011	)10011	0011001111	01111		
👌 (cary_select_adder/cin	SKO										
₽∲ (carry_select_adder/sum	011000110001011	0000111111	11111	1000111111	1000	0011000100	)1000	0110001100	01100	0110001100	01011
👌 (cary_select_adder(c	SKO										
ed lany_select_adder/c	11110			001111		111111					

#### (a)Ripple-Carry Adder

#### (b) Carry-Select Adder

Messages											
📲 🔶 /carry_skip/A	0101101110111010	0011001001	00100	0010101110	101010	0101101110	101010	01011010010	11010	0101101110	111010
₽-� /carry_skip/B	0011011001101111	00001101100	11011	0000101001	011011	0011011001	001111	00110111101	.01111	0011011001	101111
🔶 /carry_skip/Cin	9:0										
₽-� /carry_skip/Sum	1001001000101001	00111111111	11111	0011011000	000110	1001000111	11010	1001001000	101001	1001001000	101001
/carry_skip/Cout	90										
+			00000	ptco101000	001010	<u>,0001001000</u>	101010		10101	0001001000	101010
E-Y (carry_skp)p			11111			0110110111		UI1UI1UI11 6	10101		010101
Icarry_skip/c	91			000101111	111011	,011111000	,01111				
/carry_skip/X2	91										
🔶 /carry_skip/X3	9:1										

(c) Carry-Skip Adder

ISSN: 2249-6645

		A. 1 .	1. 1. 1. 1. A. 1.							
₽-1	> /Kogge_Stone/a	0000110101100100	000000001100100		000011	101100100	010011010	100100	1100110101	010100
₽-1	/Kogge_Stone/b	0010100001100100	000000001100100		001010	001100100	001010110	100100	0010101101	011000
8-1	> /Xogge_Stone/sum	0011010111001001	000000011001000	000000011001001	001101	111001001	011110001	1001000	1111100010	101100
	> /Xogge_Stone/cin	St1								
	/Kogge_Stone/cout	St0								
Ð-4	/Kogge_Stone/g	0000100001100100	000000001100100		0000100	001100100	000010010	100100	0000100101	010000
<del>8</del> -4	/Kogge_Stone/p	0010010100000000	000000000000000000000000000000000000000		001001	100000000	011001100	00000	1110011000	001100
<del>8</del> -4	/Kogge_Stone/G	000100001100100	000000001100100		000100	01100100	000111101	100100	0001111010	10000
<del>8</del> -4	/Kogge_Stone/I	0000100001100100	000000001100100		000010	001100100	000010110	100100	0000101101	010000
₽-4	/Kogge_Stone/m	000000000000000	0000000000000000000				010001000	10000	1100010000	00100
₽-4	/Kogge_Stone/q	000010000110010	00000000110010		000010	00110010	000011110	10010	0000111101	01000
₽-4	hogge_Stone/r	000000000000	000000000000							
<b>8-</b> 4	hogge_Stone/s	00000100001100	00000000001100			0001100	000001111	1100	0000011110	1010
8-4	/Kogge_Stone/t	000000000	000000000							
8-4	/Kogge_Stone/v	000010000	00000000		00000	00	000011110			

(d) Kogge-Stone Adder

Nessages						
🖬 🎝  Sparse_Kogge/a	0101110101011000	1100101110011000	0100101110011000	0101110100011000		01011101010101000
📲 🎝 /Sparse_Kogge/b	0011010010110111	0011101100110111	0011101001110111	0011111010110111		0011010010110111
🔶 /Sparse_Kogge/cin	50					
∎-ϟ  Sparse_Kogge S	1001001000001111	0000011011001111	1000011000000000	1001101111000000	1001101111001111	1001001000001111
∎-∲  Sparse_Kogge C	0111110111110000	1111101100110000	01111011111111111	011111000011111	0111110000110000	D111110111110000
📲 🍫 /Sparse_Kogge/g	010000010000	101100010000	101000010000	110000010000		<u>010000010000</u>
<b>⊪-∲</b> /Sparse_Kogge/p	100111101111	000010101111	000111101111	001110101111		100111101111
🖬 🎝 /Sparse_Kogge/G	11110100100	10100110100	11110110100	10100100100		11110100100
📲 🍫  Sparse_Kogge P	00001001011	00001000011	00001001011	00001010011		00001001011
∎-∲ (Sparse_Kogge)X	110	100	110	100		110

(e) Sparse Kogge-Stone Adder

Messages							
📲 🔶 /Sparning_Tree/a	00011011010101010	0001001001110110			0001111001010110	000111110110110	00011011010101010
🛃 🕂 (Spanning_Tree)b	0001100101111011	0001101100011011			0001101110111011	0001100101111011	
🔶 /Spanning_Tree/cin	9:1						
🚽 /Spanning_Tree/SUM	0011010011010010	0010110110010001	0010110110	010010	0011101000010010	0011100100110001	0011010011010010
🛃 🖓 (Spanning_Tree) C	0001101101111111	0001001001111110	0001001001	11111	0001111111111111	0001111111111110	000110110111111
📲 🔶 (Spanning_Tree)gt	100101010010	001000010010			101000010010	100100110010	100101010010
📲 🔶 (Spanning_Tree) pt	001000101101	100101101101			010111101101	011011001101	001000101101
🖬 🔶 (Spanning_Tree) G	1101110101	0001010101			1111110101		1101110101
📲 🔶 (Spanning_Tree)P	000000010	0000000000			000001010		000000010
🔶 /Spanning_Tree/X4	9:1						
🔶 /Spanning_Tree/X8	50						
/Spanning_Tree/X12	9:1						

#### (f) Spanning Tree adder

Figure: (a)-(f): A 16-bit parallel prefix adder simulation result for all combinations outputs.

For the HDL structural design, the test vectors for excitation has been provided, and the response is as shown in Figure. Here the input reference vector is a=0010110111010101, b=0010110011011101001, for Ripple carry adder,

a=0010111100111100, b=0011001111001111, for Carry select adder, a=010110111011101,b=0011011001101111 for Carry skip adder.

a=0000110101100100,b=0010100001100100 for Kogge stone adder,

a=01011101010100,b=0011010010110111 for sparse kogge stone adder,

a=0001101101010110,b=0001100101111011 for panning tree adder.

## VI. SYNTHESIS REPORT

Final <i>Results</i>	
RTL Top Level Output File	Name : ripple carry adder.ng
Top Level Output File Name	: ripple carry adder
Output Format	: NGC
Optimization Goal	: Speed
Keep Hierarchy	: No
Design Statistics	
# IOs : 50	

	- · · · · · · · · · · · · · · · · · · ·	
#	BELS	: 32
#	LUT3	: 32
#	IO Buffers	: 50
#	IBUF	: 33
#	OBUF	: 17

#### Timing constraints

21.69ns (Levels of Logic = 18) Delay: Source: B < 0 > (PAD)Destination: C out (PAD)

Data Path: B<0> to C out

Cell:	Fan	Gate delay	Net delay	Logic
In_>Out	out			Name(Net
		1.10.6	0.530	Name)
IBUF:I- >O	2	1.106	0.532	B_0_IBUF (B_0_IBUF)
LUT3:10-	2	0.612	0.449	FA0/cout1 (c<0>)
LUT3:I1- >0	2	0.612	0.449	FA1/cout1 (c<1>)
LUT3:I1- >0	2	0.612	0.449	FA2/cout1 (c<2>)
LUT3:I1- >0	2	0.612	0.449	FA3/cout1 (c<3>)
LUT3:I1- >0	2	0.612	0.449	FA4/cout1 (c<4>)
LUT3:I1- >0	2	0.612	0.449	FA5/cout1 (c<5>)
LUT3:I1- >0	2	0.612	0.449	FA6/cout1 (c<6>)
LUT3:I1- >0	2	0.612	0.449	FA7/cout1 (c<7>)
LUT3:I1- >0	2	0.612	0.449	FA8/cout1 (c<8>)
LUT3:I1- >0	2	0.612	0.449	FA9/cout1 (c<9>)
LUT3:I1- >0	2	0.612	0.449	FA10/cout1 (c<10>)
LUT3:I1- >0	2	0.612	0.449	FA11/cout1 (c<11>)
LUT3:I1- >0	2	0.612	0.449	FA12/cout1 (c<12>)
LUT3:I1- >0	2	0.612	0.449	FA13/cout1 (c<13>)
LUT3:I1->0	2	0.612	0.449	FA14/cout1 (c<14>)
LUT3:I1- >0	1	0.612	0.357	FA15/cout1 (c<15>)
OBUF:I- >O		3.169		Cout_ OBUF (Cout)

#### Final Results

RTL Top Level Output File Name	: kogge-stone adder.ngr
Top Level Output File Name	: kogge-tone adder
Output Format	: NGC
Optimization Goal	: Speed
Keep Hierarchy	: No

# Design Statistics

: 50

### Cell Usage:

# IOs

	0	
#	BELS	: 41
#	GND	:01
#	LUT3	: 27
#	LUT4	:9
#	IO Buffers	: 50
#	IBUF	: 33
#	OBUF	: 17

## Timing constraints

	T.	C.A.	NL 4	
<i>a</i>	Fan	Gate	Net	Logic name(Net Name)
Cell: 1n-	out	delay	delay	
>out				
IBUF:I-	4	1.106	0.651	b_1_IBUF (b_1_IBUF)
>0				,
	1	0.612	0.000	GC2/G1 SW01
I I/T/4.10	1	0.012	0.000	$(CC2/C1_5)(01)$
LU14:10-				$(GC2/G1_5W0)$
>0				
MUXF5:I1-	2	0.278	0.410	GC2/G1_SW0_f5
>0				(q<1>)
LUT3:12-	2	0.612	0.532	GC2/G1 (q<2>)
>0	-	01012	01002	001/01 (q)
	2	0 (12	0.522	CCCCC CWA CWA
LU15:10-	2	0.012	0.552	GC0/G_8W0_8W0
>0				(s<3>)
LUT3:I0-	2	0.612	0.532	GC7/G_SW0_SW0
>0				(s<4>)
LUT3:10-	2	0.612	0.532	GC8/G SW0 SW0
>0	-	01012	01202	(\$<5>)
	2	0 (12	0.410	
LU13:10-	2	0.612	0.410	GC9/G_8W0_8W0
>0				(s<6>)
LUT3:I2-	3	0.612	0.603	GC9/G_SW0 (v<7>)
>0				
	2	0.612	0.410	GC9/G_SW1 (v<8>)
1.1173.10	-	0.012	0.110	
1015.10-				
>0	_			
	2	0.612	0.410	GC9/G (v<9>)
LUT3:I2-				
>0				
	2	0.612	0.532	GC12/G SW0 (v<10>)
LUT3.12.	_			0011,010,000,000,000,000,000,000,000,00
>0				
20		0.440	0.44.0	
	2	0.612	0.410	GCI2/G_SWI
LUT3:10-				(GC13/G5)
>0				
	2	0.612	0.410	GC12/G (GC14/G9)
LUT3:12-				
NO 10.12				
~0	•	0 (12	0.410	0.014/019 (0.012/024)
	2	0.612	0.410	GC14/G18 (GC13/G34)
LUT3:12-				
>0				
	1	0.612	0.357	Mxor sum<14> Result1
LUT3:12-				(sum 14 OBUF)
50				(50001)
~0		2.1(0		and 14 ODUE
0.001/0.1		3.109		sum_14_OBUF
OBUF:1-				(sum<14>)
>0				1

#### Final Results

RTL Top Level Output File Name: sparse kogge-stone Adder.ngr Top Level Output File Name : sparse kogge Output Format : NGC Optimization Goal : Speed Keep Hierarchy : No **Design** Statistics # IOs : 65 Cell Usage: # BELS : 54 # LUT2 :02 : 30 # LUT3 # LUT4 :19

		/
#	MUXF5	:03
#	IO Buffers	: 65
#	IBUF	: 33
#	OBUF	: 32

#### Timing constraints

Delay:15.916ns (Levels of Logic = 13)Source:a < 6 > (PAD)Destination:C < 6 > t (PAD)Data Path:a < 6 > to C < 16 >

Cell:	Fan	Gate	Net	Logic
in >out	out	delav	delav	Name(Net
				Name)
IBUF:I-	4	1.106	0.651	a 6 IBUF
>0				(a 6 IBUF)
	2	0.612	0.449	BC8/G18
LUT4:10-				(BC8/G18)
>0				(_ 00: 010)
	1	0.612	0.000	BC8/G461
LUT4:I1-				(BC8/G461)
>0				(_ 00: 01:02)
	3	0.278	0.603	BC8/G46_f5
MUXE5:11-	-			(BC8/G46)
>0				(200/010)
	1	0.612	0.387	GC3/C13
LUT4:10-	-	0.012	0.007	(GC3/C13)
>0				(000,010)
70	1	0.612	0.360	GC3/C21
LUT3.12.	-	0.012	0.200	(GC3/C21)
>0				(003/021)
	1	0.612	0.426	GC3/C46
LUT4-13-	-	0.012	0.120	(GC3/C46)
>0				(005/040)
70	2	0.612	0 449	GC3/C77
I UT4.I1.	-	0.012	0.112	(GC3/C77)
NO				(005/077)
20	3	0.612	0.520	FA13/cout1
I UT3.II	3	0.012	0.320	(C 13  OBUE)
LU15.11-				(C_13_0B0F)
~0	3	0.612	0.520	FA14/cout1
I IIT2.II	3	0.012	0.320	(C 14  OPUE)
				$(C_14_0D0F)$
~0	2	0.612	0.520	EA15/cont1
I UT2.11	3	0.012	0.520	FA15/COULI
LU13:11-				(C_15_OBUF)
>0	1	0.(10	0.257	TAICLE 11
1 1/02 11	1	0.612	0.357	FA16/couti
LU13:11-				(C_16_OBUF)
>0		2.1.0		C 44 ODVE
OBUF:I-		3.169		C_16_OBUF
>0	1		1	(C<16>)

### VII. IMPLEMENTATION AND RESULTS

The proposed design is functionally verified and the results are verified. The timing report was obtained. The Simulation Verified in Modelsim and Synthesis was verified in Xilinx.

International Journal of Modern Engineering Research (IJMER)

www.ijmer.com

Vol.	Vol. 3, Issue. 5, Sep - Oct. 2013 pp-3107-3115			
N Delay		Delay	Delay	
	trca	<i>tKs</i>	ts <b>k</b> s	
16	21.690ns	20.262ns	15.916ns	

ISSN: 2249-6645

#### VIII. CONCLUSION

In this paper An improved optimization techniques for parallel prefix adder has been proposed and implemented. The design of the proposed prefix adders is done using Ripple carry adder and Kogge-stone adder, Sparse kogge tone adder and panning tree adder. speed of parallel prefix adder is increased compared to the Ripple carry adder. The functional verification of the proposed design of the An improved optimization techniques for parallel prefix adder is performed through simulations using the Verilog HDL flow in ModelSim for prefix adders and Synthesis done using Xilixn. The design of An improved optimization technique for parallel prefix adder can performed. The proposed design of An improved optimization techniques for parallel prefix adder can perform ripple carry adder, kogge stone adder, spare kogge adder, spanning tree adder , parallel adder give the better result compared to the ripple carry adder.

#### 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. 37<sup>th</sup> Asiloma r 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.