International Journal of Modern Engineering Research (IJMER)
www.ijmer.com Vol. 3, Issue. 5, Sep - Oct. 2013 pp-3107-3115 ISSN: 2249-6645

An Improved Optimization Techniques for Parallel Prefix Adder using FPGA

O.Anjaneyulu[1], L.Swapna[2], C.V.Krishna Reddy[3],
Member, IEEE KITS,Warangal[12], NNRESGI-Hyd[3] 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 (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].

$$(G_L, P_L) \square (G_R, P_R) = (G_L + P_L \bullet G_R, P_L \bullet P_R) \quad (1)$$

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:

$$c_4 = (g_4, p_4) \square [(g_3, p_3) \square [(g_2, p_2) \square (g_1, p_1)]] \quad (2)$$

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

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

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.

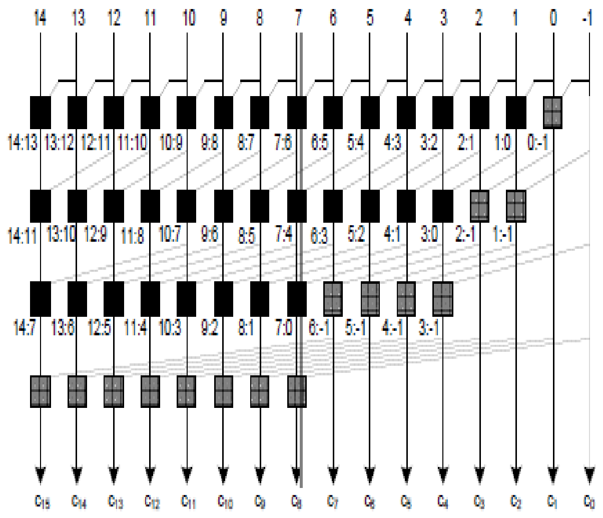


Fig1.16 bit Kogge-Stone adder

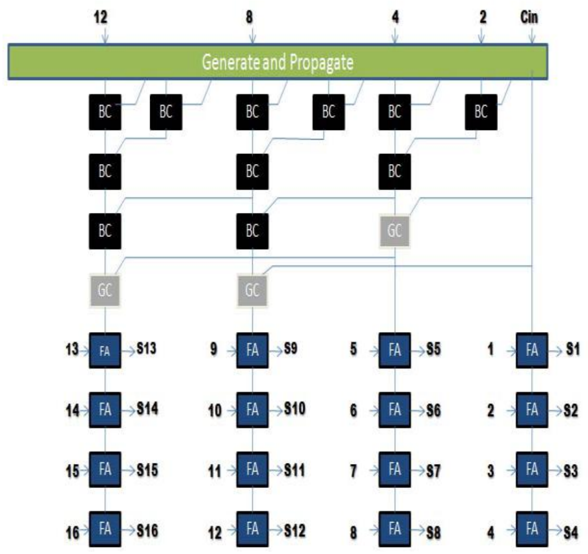


Fig2. sparse 16 bit Kogge-Stone adder

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].

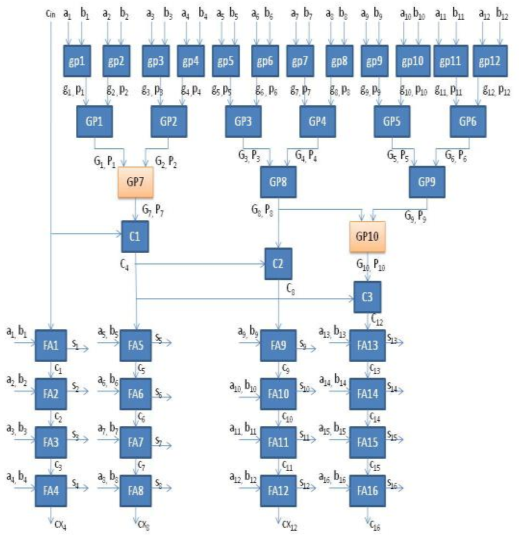


Fig3. Spanning Tree Carry Look ahead Adder (16 bit)

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.

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

(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)**

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. 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**KS* **= (*n*+2)□ □L U T + □ □ (*n*) (4)** where $N = 2n$, the adder bit width, ΔLUT is the delay through a lookup table (LUT), and ρKS(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)□ □ □U X + □R CA (5)**
where ΔMUX is the mux delay associated with the fast-carry chain and τR 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: ΔLUT = 0.612 ns, ΔMUX = 0.051 ns, and τR C A = 1.715 ns. Even though ΔMUX << ΔLUT, it is expected that the Kogge-Stone adder will eventually be faster than the RCA because $N = 2n$, provided that ρKS(n) grows relatively slower than (N – 2)□ΔMUX. Indeed, Table II predicts that the Kogge-Stone adder will have superior performance at N =256.

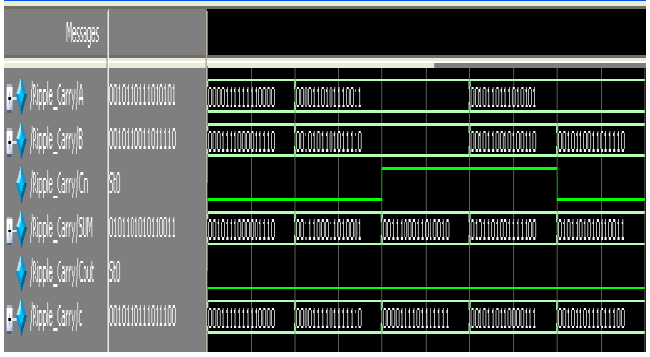
Table2 : Delay Results for the Kogge-Stone Adders

N	**Synth. Predict**	**Route Delay**	**Route Fitted**	**Delay *tKS***	**Delay *tRCA***
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**

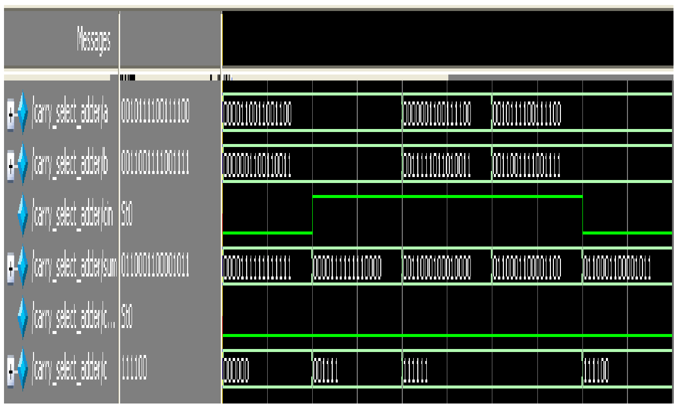
(all delays given in ns)

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 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.

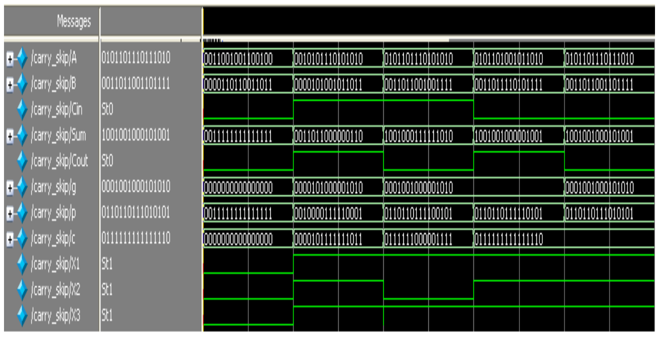
V. Simulation Results



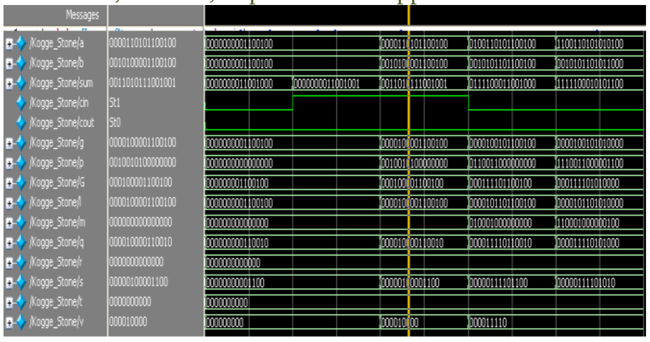
(a)Ripple-Carry Adder



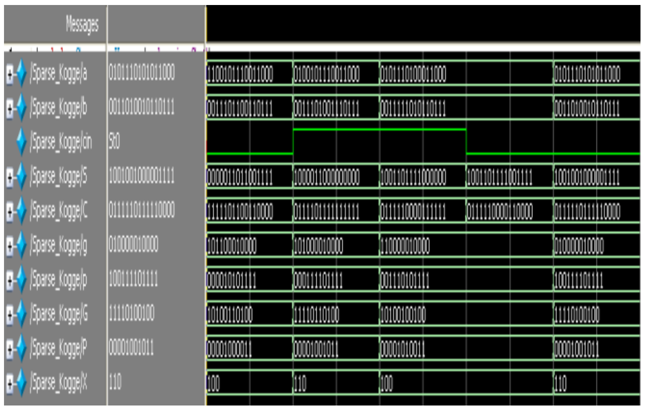
(b) Carry-Select Adder



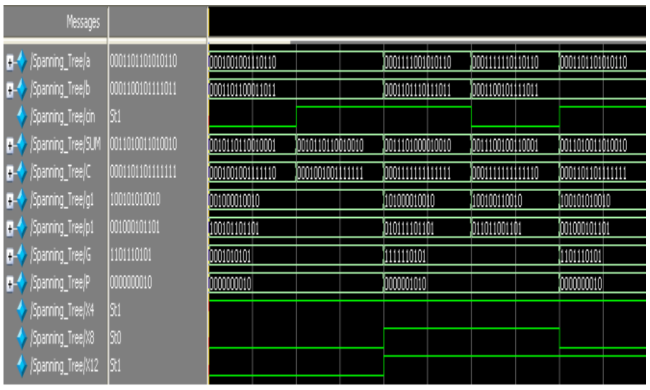
(c) Carry-Skip Adder



(d) Kogge-Stone Adder



(e) Sparse Kogge-Stone Adder



(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=0010110011011110,for Ripple carry adder, a=0010111100111100, b=0011001111001111, for Carry select adder, a=0101101110111010,b=0011011001101111 for Carry skip adder.
a=0000110101100100,b=0010100001100100 for Kogge stone adder,
a=0101110101011000,b=0011010010110111 for sparse kogge stone adder,
a=0001101101010110,b=0001100101111011 for panning tree adder.

VI. SYNTHESIS REPORT

Final *Results*

RTL Top Level Output File Name	: ripple carry adder.ngr
Top Level Output File Name	: ripple carry adder
Output Format	: NGC
Optimization Goal	: Speed
Keep Hierarchy	: No

Design Statistics

# IOs	: 50

Cell Usage :

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

Timing constraints

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

Cell: In_>Out	**Fan out**	**Gate delay**	**Net delay**	**Logic Name(Net Name)**
IBUF:I->O	**2**	**1.106**	**0.532**	**B_0_IBUF (B_0_IBUF)**
LUT3:I0->O	**2**	**0.612**	**0.449**	**FA0/cout1 (c<0>)**
LUT3:I1->O	**2**	**0.612**	**0.449**	**FA1/cout1 (c<1>)**
LUT3:I1->O	**2**	**0.612**	**0.449**	**FA2/cout1 (c<2>)**
LUT3:I1->O	**2**	**0.612**	**0.449**	**FA3/cout1 (c<3>)**
LUT3:I1->O	**2**	**0.612**	**0.449**	**FA4/cout1 (c<4>)**
LUT3:I1->O	**2**	**0.612**	**0.449**	**FA5/cout1 (c<5>)**
LUT3:I1->O	**2**	**0.612**	**0.449**	**FA6/cout1 (c<6>)**
LUT3:I1->O	**2**	**0.612**	**0.449**	**FA7/cout1 (c<7>)**
LUT3:I1->O	**2**	**0.612**	**0.449**	**FA8/cout1 (c<8>)**
LUT3:I1->O	**2**	**0.612**	**0.449**	**FA9/cout1 (c<9>)**
LUT3:I1->O	**2**	**0.612**	**0.449**	**FA10/cout1 (c<10>)**
LUT3:I1->O	**2**	**0.612**	**0.449**	**FA11/cout1 (c<11>)**
LUT3:I1->O	**2**	**0.612**	**0.449**	**FA12/cout1 (c<12>)**
LUT3:I->O	**2**	**0.612**	**0.449**	**FA13/cout1 (c<13>)**
LUT3:I1->O	**2**	**0.612**	**0.449**	**FA14/cout1 (c<14>)**
LUT3:I1->O	**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

# IOs	: 50

Cell *Usage*:

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

Timing constraints

Delay: 20.262ns (Levels of Logic = 17)
Source: b<1> (PAD)
Destination: Sum<14> (PAD)
Data Path: b<1> to Sum<14>

Cell: in->out	**Fan out**	**Gate delay**	**Net delay**	**Logic name(Net Name)**
IBUF:I->O	**4**	**1.106**	**0.651**	**b_1_IBUF (b_1_IBUF)**
LUT4:I0->O	**1**	**0.612**	**0.000**	**GC2/G1_SW01 (GC2/G1_SW0)**
MUXF5:I1->O	**2**	**0.278**	**0.410**	**GC2/G1_SW0_f5 (q<1>)**
LUT3:I2->O	**2**	**0.612**	**0.532**	**GC2/G1 (q<2>)**
LUT3:I0->O	**2**	**0.612**	**0.532**	**GC6/G_SW0_SW0 (s<3>)**
LUT3:I0->O	**2**	**0.612**	**0.532**	**GC7/G_SW0_SW0 (s<4>)**
LUT3:I0->O	**2**	**0.612**	**0.532**	**GC8/G_SW0_SW0 (s<5>)**
LUT3:I0->O	**2**	**0.612**	**0.410**	**GC9/G_SW0_SW0 (s<6>)**
LUT3:I2->O	**3**	**0.612**	**0.603**	**GC9/G_SW0 (v<7>)**
LUT3:I0->O	**2**	**0.612**	**0.410**	**GC9/G_SW1 (v<8>)**
LUT3:I2->O	**2**	**0.612**	**0.410**	**GC9/G (v<9>)**
LUT3:I2->O	**2**	**0.612**	**0.532**	**GC12/G_SW0 (v<10>)**
LUT3:I0->O	**2**	**0.612**	**0.410**	**GC12/G_SW1 (GC13/G5)**
LUT3:I2->O	**2**	**0.612**	**0.410**	**GC12/G (GC14/G9)**
LUT3:I2->O	**2**	**0.612**	**0.410**	**GC14/G18 (GC13/G34)**
LUT3:I2->O	**1**	**0.612**	**0.357**	**Mxor_sum<14>_Result1 (sum_14_OBUF)**
OBUF:I->O		**3.169**		**sum_14_OBUF (sum<14>)**

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
#	LUT3	: 30
#	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: in_>out	Fan out	Gate delay	Net delay	Logic Name(Net Name)
IBUF:I->O	4	1.106	0.651	a_6_IBUF (a_6_IBUF)
LUT4:I0->O	2	0.612	0.449	BC8/G18 (BC8/G18)
LUT4:I1->O	1	0.612	0.000	BC8/G461 (BC8/G461)
MUXF5:I1->O	3	0.278	0.603	BC8/G46_f5 (BC8/G46)
LUT4:I0->O	1	0.612	0.387	GC3/C13 (GC3/C13)
LUT3:I2->O	1	0.612	0.360	GC3/C21 (GC3/C21)
LUT4:I3->O	1	0.612	0.426	GC3/C46 (GC3/C46)
LUT4:I1->O	2	0.612	0.449	GC3/C77 (GC3/C77)
LUT3:I1->O	3	0.612	0.520	FA13/cout1 (C_13_OBUF)
LUT3:I1->O	3	0.612	0.520	FA14/cout1 (C_14_OBUF)
LUT3:I1->O	3	0.612	0.520	FA15/cout1 (C_15_OBUF)
LUT3:I1->O	1	0.612	0.357	FA16/cout1 (C_16_OBUF)
OBUF:I->O		3.169		C_16_OBUF (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.

N	Delay t_{RCA}	Delay t_{KS}	Delay t_{SKS}
16	**21.690ns**	**20.262ns**	**15.916ns**

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 has been 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. 37th 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.