IOSR Journal of VLSI and Signal Processing (IOSR-JVSP)
ISSN: 2319 – 4200, ISBN No. : 2319 – 4197 Volume 1, Issue 1 (Sep-Oct. 2012), PP 51-57
www.iosrjournals.org

Implementation and Performance Evaluation of Prefix Adders using FPGAs

K.Babulu[1], Y.Gowthami[2]
[1] *Professor of ECE, UCEK, JNTUK, Kakinada, India.*
[2] *Project Associate, ECE Department, UCEK, JNTUK, Kakinada, India.*

ABSTRACT: *Parallel Prefix Adders have been established as the most efficient circuits for binary addition. The binary adder is the critical element in most digital circuit designs including digital signal processors and microprocessor data path units. The final carry is generated ahead to the generation of the sum which leads extensive research focused on reduction in circuit complexity and power consumption of the adder. In VLSI implementation, parallel-prefix adders are known to have the best performance. This paper investigates four types of carry-tree adders (the Kogge-Stone, sparse Kogge-Stone, spanning tree, Brent kung Adder) and compare them to the simple Ripple Carry Adder and Carry Skip Adder. These designs of varied bit-widths are simulated using model-sim simulator of 6.4 version and implemented on a Xilinx 10.1 version Spartan 3E FPGA. These fast carry-chain carry-tree adders support the bit width up to 256. We report on the area requirements and reduction in circuit complexity for a variety of classical parallel prefix adder structures.*

Keywords*-FPGA, Prefix Adder, ALU, Xilinx, VLSI, CLA, Simulation, Synthesis*

I. INTRODUCTION

Binary addition is a fundamental operation in most digital circuits. There are a variety of adders, each has certain performance. Each type of adder is selected depending on where the adder is to be used. Adders are critically important elements in processor chips and they are used in floating-point arithmetic units, ALUs, memory addressing, program counter updating, Booth Multipliers, ALU Designing, multimedia and communication systems, Real-time signal processing like audio signal processing, video/image processing, or large capacity data processing etc. The requirements of the adder are that it is primarily fast and secondarily efficient in terms of power consumption. In VLSI implementations, parallel-prefix adders are known to have the best performance. In this paper, designing and implementing the tree-based adders on FPGAs are described. Tree-based adder structures are implemented on FPGA and compared with the Ripple Carry Adder (RCA) and the Carry Skip Adder (CSA). Some conclusions and suggestions are made for improving FPGA designs to enable better tree-based adder performance.

Parallel prefix (or tree prefix) adders provide a good theoretical basis to make a wide range of design trade-offs in terms of delay, area and power. Parallel Prefix Adders (PPA) is designed by considering carry look adder as a base. Similar to a CLA they employ the 3-stage structure shown in Figure.1 CLA and a PPA differ in second stage. In second stage carry signal of binary addition is generated.

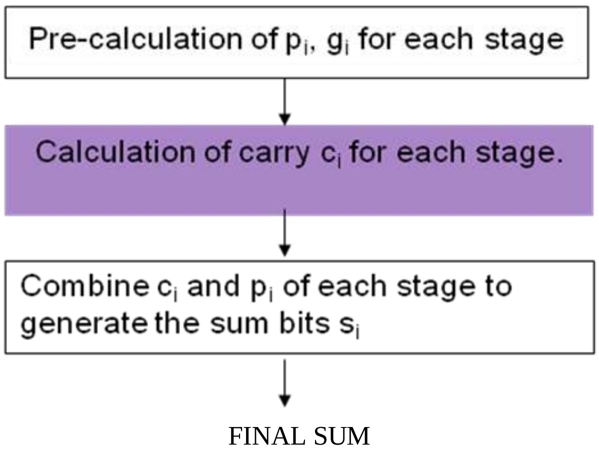

Pre-calculation of p_i, g_i for each stage
Calculation of carry c_i for each stage.
Combine c_i and p_i of each stage to generate the sum bits s_i
FINAL SUM

Figure.1 Stages of Binary Addition

Three stage structure of the carry look ahead and parallel prefix adder. In a PPA the prefix operator "o" is introduced and the carry signal generation is treated as a prefix problem [1].The operator is defined by the equation:

$$(G_i, P_i) = (g_i, p_i) \circ (G_{i-1}, P_{i-1}) \quad - [1]$$

Where:

$$(G_0, P_0) = (g_0, p_0)$$

$$(g_x, p_x) \circ (g_y, p_y) = (g_x + p_x \cdot g_y, p_x \cdot p_y)$$

with $+, \cdot$ ***being the OR, AND operations***

In equation [1] "o" is applied on (gin1, pin1) and (gin2, pin2). These pairs represent generate and propagate signals used in the addition. The output of the operator is a new pair of bits generated as described in equation (1). The adders in third are built from generate and propagate (GP) blocks, black cells (BC) blocks, eight gray cell (GC) blocks. The details of the various blocks used in the structure adders mentioned in this are discussed below:

1. GP block

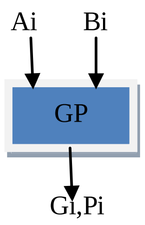
The generate and propagate block takes a pair of operand bits (a, b) as inputs. Computes a pair of generate and propagate signals (g, p) as output. Generate (Gi) indicates whether a carry is generated from that bit

Gi = Ai & Bi

Propagate (Pi) indicates whether a carry is propagated through that bit

Pi = Ai xor Bi

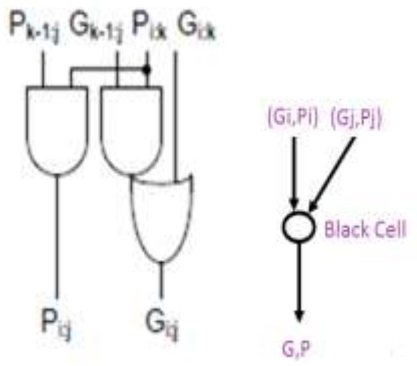
Generate Propagate Block



2. BC block

The black cell takes two pairs of generate and propagate signals (gi, pi) and (gj, pj) as input. computes a pair of generate and propagate signals (g, p) as output

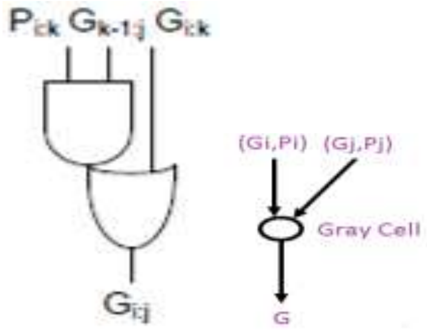
Gi.j = Gi + (Pi . Gj)
Pi.j = Pi . Pj



Logic Diagram

3. GC block:

The gray cell takes two pairs of generate and propagate signals (Gi, Pi) and (Gj, Pj) as inputs. Computes a generate signal "G "
as output Gi,j = Gi + (Pi . Gj



Logic Diagram

These are the cell definitions and there logic diagrams for generate and propagate. Black Cell consists of both Generate and propagate where as Gray Cell consists of Generate. The arrangement of the prefix network gives rise to various families of adders.

II. RELATED WORK

We compared the design of the ripple carry adder with the carry-look ahead, carry-skip, and carry-select adders on the Xilinx 4000 series FPGAs. Only an optimized form of the carry-skip adder performed better than the ripple carry adder when the adder operands were above 56 bits. A study of adders implemented on the Xilinx Vertex II yielded similar results. The previous authors considered several parallel prefix adders implemented on a Xilinx Vertex 5 FPGA. It is found that the simple RCA adder is superior to the parallel prefix designs because the RCA can take advantage of the fast carry chain on the FPGA.

This study focuses on carry-tree adders implemented on a Xilinx Spartan 3E FPGA. The distinctive contributions of this paper are two-fold. First, we consider tree-based adders and a hybrid form which combines a tree structure with a ripple-carry design. The Kogge-Stone adder is chosen as a representative of the former type and the sparse Kogge Stone and Brent Kung Adder is representative of the latter category. Second, this paper considers the practical issues involved in testing the adders and provides actual measurement data to compare with simulation results. The previous works cited above all rely upon the synthesis reports from the FPGA place and route software for their results.

A 16-bit Kogge-Stone adder is built from 16 generate and propagate (GP) blocks, 37 black cells (BC) blocks, 16 (GC) blocks, 16 sum blocks. Kogge-Stone prefix tree is one of the adders that use fewest logic levels. Gray cells are inserted similar to black cells except that the gray cells final output carry outs instead of intermediate G/P group. The reason of starting with Kogge-Stone prefix tree is that it is the easiest to build in terms of using a program concept. The Figure.2 shown below is 16-bit (a power of 2) prefix tree and it is not difficult to extend the structure to any width if the basics are strictly followed.

The sparse Kogge-Stone adder consists of several smaller ripple carry adders (RCAs) on its lower half, a carry tree on its upper half. It terminates with RCAs. The number of carries generated is less in a sparse Kogge-Stone adder compared to the regular Kogge-Stone adder. The functionality of the GP block, black cell and the gray cell remains exactly the same as in the regular Kogge-Stone adder. 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. The Figure.3 Shown below is the Block diagram of 16-Bit Sparse Kogge-Stone Adder.

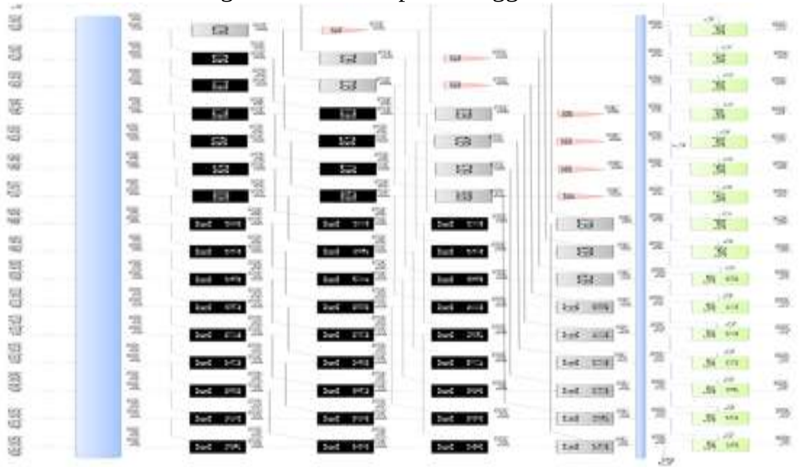


Figure.2 16-Bit Kogge-Stone Adder

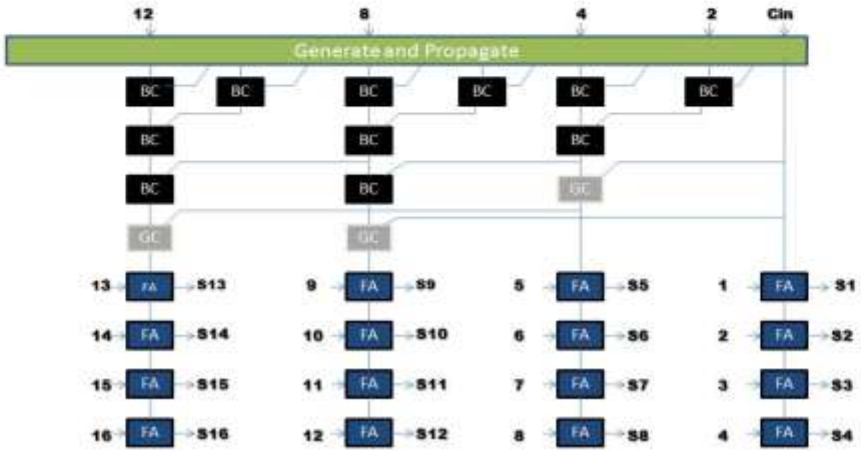


Figure.3 Block Diagram of Sparse Kogge Stone adder

III. BRENT KUNG ADDER

The Brent Kung adder is the advanced version of Kogge stone adder. Kogge Stone adder generates intermediate carries shown in the block diagram is very attractive for high-performance applications. However, it comes at the cost of area and power. A simpler tree structure could be formed if only the carry at every power of two positions is computed as proposed by Brent and Kung [7]. Figure.4 shows a 16-bit prefix tree of their idea. An inverse carry tree is added to compute intermediate carries. Its wire complexity is much less than that of Kogge Stone adder. The number of slices, LUT, IOB Bounds is less compared to Kogge Stone Adder, Sparse Kogge stone Adder and Spanning Tree Carry Look Adder. When the usage of components reduces, the circuit complexity reduces, cost also gets reduces. so according to application either of the adder are used.

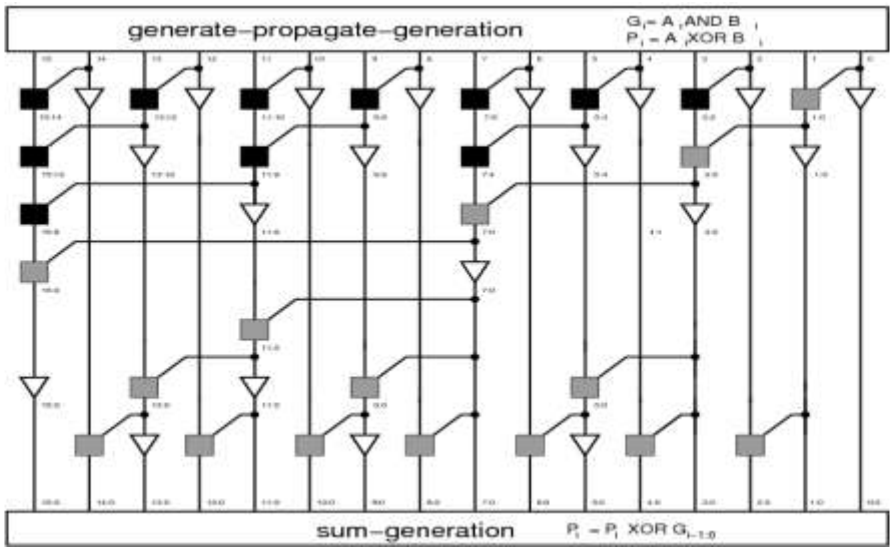


Figure.4 Block Diagram of 16-Bit Brent Kung Adder

IV. METHODOLOGY

The adders to be studied were designed with varied bit widths up to 256 bits and coded in VERILOG. The functionality of the adders were verified by using Model-sim Simulator. The Xilinx ISE 10.1 software was used to synthesize the designs onto the Spartan 3E FPGA. 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.

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

(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)

Table.1 Subset of (g, p) Relations used for Testing

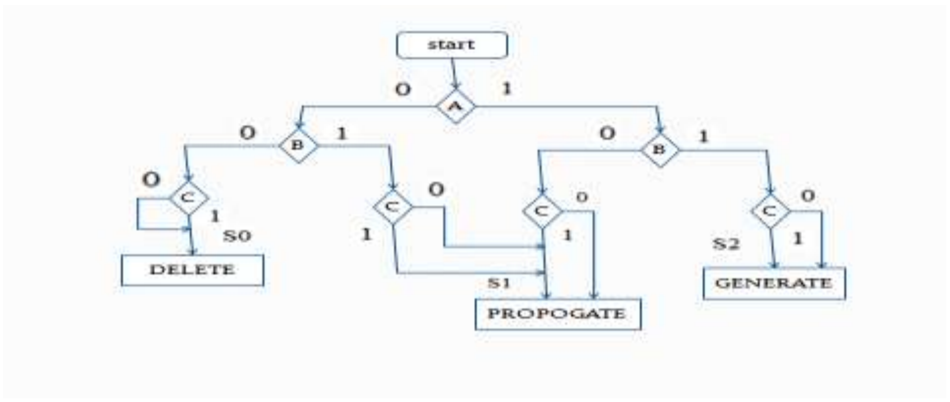


Figure.5 ASM Chart for 2- input with Carry-in.

ASM Chart consists of 3 inputs A, B, C. When AB is 00 irrespective of input carry the output state is DELETE state, when inputs are 01 or 10 output carry is generated depending on input carry so output state is considered as PROPAGATE state, when the input 11 irrespective of input carry, out carry is generated so the state is name as GENERATE.

A	B	Cin	OUTPUT STATE
0	0	0	
0	0	1	DELETE
0	1	0	
0	1	1	PROPAGATE STATE
1	0	0	
1	0	1	
1	1	0	GENERATE
1	1	1	

Table.2 Truth Table for 2-Input with carry-in

V. RESULTS

Table.3 contains the results obtained. The adder abbreviations used in the table and the following discussions are: BK for the Brent-Kung adder, KS for the Kogge-Stone adder, SK for Sparse Kogge Stone Adder, RC for Ripple Carry Adder. In the table, area is measured in Slice Look-Up Tables (LUT) units which represent configurable logic units within the FPGA. Interestingly the synthesis tool synthesized a simple ripple carry adder regardless of the optimization strategy. The adder was implemented by configuring the slices within the FPGA as full adder components.

Hence the number of lookup tables matched the operand bit-size in every case. Table.3 give the area results with the software set for area. It is apparent from these tables that the area optimization strategy produces adders which are significantly smaller compared to those produced with complexity optimization. In Table we can observe that generally the adder areas compare with the characteristics of their type. The BK exhibits the smallest size while the KS is the largest adder. This shows that in certain cases the tool optimized circuits reverse the algorithmic superiority of a design.

	RC	KS	SK	BK
I/P Arrival Time	18.43	12.72	12.65	10.82
Path delay	21.65	20.26	15.87	5.96
Slices	18	21	29	26
4 i/p LUT's	32	37	51	45
Bonded IOB's	51	51	66	56

Table.3 Area Results Obtained With Area Optimization

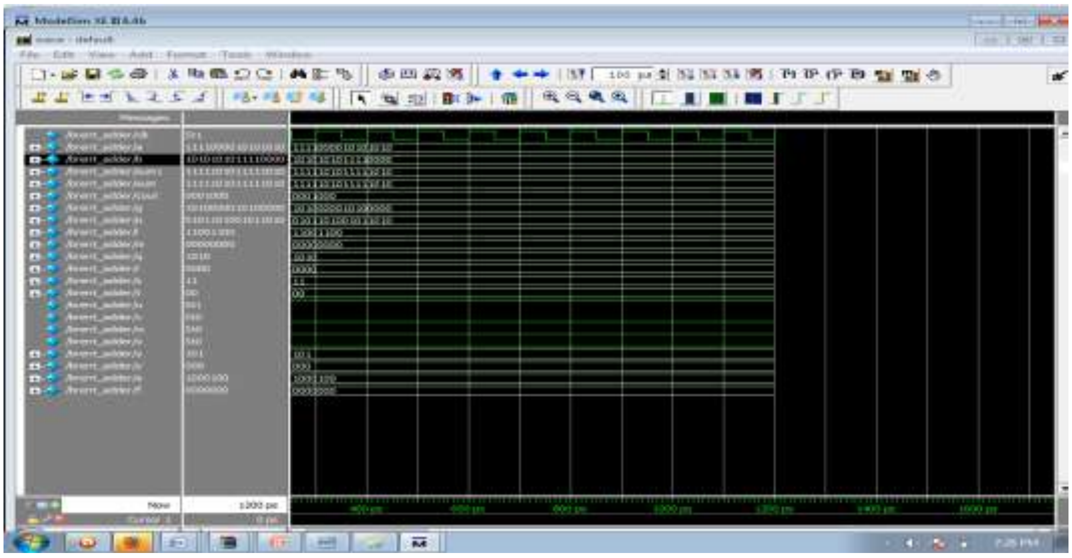


Figure.6 Simulation results for Brent Kung Adder

Figure.6 shows the Simulation results for Brent Kung Adder which includes 2 i/p of 16 bit. By using Brent Kung Adder sum and carry is generated where, it produces path delay less when compared to other adders. This waveform is simulated using Model-sim software. Figure.7 is the bar diagram representation for adders. Already existing adders takes more delay to produce the carry. this has been reduced by using Brent Kung adder.

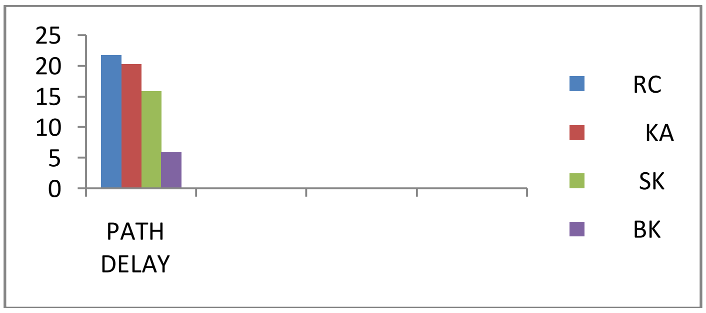


Figure.7 Comparison of path delays for Adders

VI. CONCLUSION

Parallel-prefix adders are not as effective as the simple ripple-carry adder at low to moderate bit widths. We have indications that the carry-tree adders eventually surpass the performance of the linear adder designs at high bit-widths, expected to be in the 128 to 256 bit range. This is important for large adders used in precision arithmetic and cryptographic applications where the addition of numbers on the order of a thousand bits is not uncommon. Because the adder is often the critical element which determines to a large part the cycle time and power dissipation for many digital signal processing and cryptographically implementations, it would be worthwhile for future FPGA designs to include an optimized carry path to enable tree based adder designs to be optimized for place and routing. The testability and possible fault tolerant features of the Brent kung adder are topics for future research.

REFERENCES

[1] Andrew Beaumont-Smith and Cheng-Chew Lim,Parallel Prefix Adder Design, 0-769s-I 150-3/01 $10.00 0 2001 IEEE
[2] Dinesh Patil, Omid Azizi, Mark Horowitz, Ron Ho, Rajesh Ananthraman, Robust Energy-Efficient Adder Topologies
[3] Abhijith Kini G, Asynchronous Hybrid Kogge-Stone Structure Carry Select Adder Based IEEE-754 Double-Precision Floating-Point Adder, IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 5, No 2, September 2011 ISSN (Online): 1694-0814
[4] Sabyasachi Das and Sunil P. Khatri, A Novel Hybrid Parallel-Prefix Adder Architecture With Efficient Timing-Area Characteristic, IEEE Transactions on VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, VOL. 16, NO. 3, MARCH 2008
[5] Hoang Q. Dao and Vojin G. Oklobdzija, Performance Comparison of VLSI Adders Using Logical Effort
[6] Radu Zlatanovici, Borivoje Nikolic, Power – Performance Optimal 64-Bit Carry-Lookahead Adders.
[7] Cory Merkel, David Brenner, 8-bit Parallel Prefix Adders Using Brent Kung Tree with BIST.
[8] R.W.Doran,Variants of an Improved Carry Look-Ahead Adder,IEEE Transactions on Computers,Vol.37,No9,September 1988.

Dr.K.Babulu[1] received the Ph.D. degree in VLSI from JNTUA, India in the year 2010. At present he has been working as Professor and headed with the department of ECE, UCEK, JNTUK, and Kakinada, India. His research interests include VLSI Design, Embedded System Design, ASIC Design and PLD design. He has published over 29 papers in various national and international journals and conferences.

Y.Gowthami[2] is a Project Associate pursuing her MTech Degree with Computers and Communications specialization at the department of Electronics and Communication Engineering, UCEK, JNTUK, Kakinada.