A Novel Approach For Designing A Low Power Parallel Prefix Adders

R.Chaitanyakumar
M Tech student,
Pragati Engineering College,
Surampalem (A.P, IND).

P.Sunitha
Assistant Professor, Dept.of ECE
Pragati Engineering College,
Surampalem *(A.P, IND).*

Abstract

This paper presents an implementation of prefix adders and compare with the Ripple carry adder (RCA) and Carry skip adder (CSA) in terms of power and speed. The binary adder is the critical element in most digital circuit designs including digital signal processors (DSP) and microprocessor data path units. As such, extensive research continues to be focused on improving the power delay performance of the adder. In VLSI implementations, parallel-prefix adders are known to have the best performance.

Binary adders are one of the most essential logic elements within a digital system. In addition, binary adders are also helpful in units other than Arithmetic Logic Units (ALU), such as multipliers, dividers and memory addressing. Therefore, binary addition is essential that any improvement in binary addition can result in a performance boost for any computing system and, hence, help improve the performance of the entire system. In this project Xilinx-ISE tool is used for simulation, logical verification, and further synthesizing.

1. Introduction

To humans, decimal numbers are easy to comprehend and implement for performing arithmetic. However, in digital systems, such as a microprocessor, DSP (Digital Signal Processor) or ASIC (Application-Specific Integrated Circuit), binary numbers are more pragmatic for a given computation. This occurs because binary values are optimally efficient at representing many values.

Binary adders are one of the most essential logic elements within a digital system. In addition, binary adders are also helpful in units other than Arithmetic Logic Units (ALU), such as multipliers, dividers and memory addressing. Therefore, binary addition is essential that any improvement in binary addition can Result in a performance boost for any computing system and, hence, help improve the performance of the entire system.

The major problem for binary addition is the carry chain. As the width of the input operand increases, the length of the carry chain increases. Figure 1.1 demonstrates an example of an 8- bit binary add operation and how the carry chain is affected. This example shows that the worst case occurs when the carry travels the longest possible path, from the least significant bit (LSB) to the most significant bit (MSB). In order to improve the performance of carry-propagate adders, it is possible to accelerate the carry chain, but not eliminate it. Consequently, most digital designers often resort to building faster adders when optimizing computer architecture, because they tend to set the critical path for most computations.

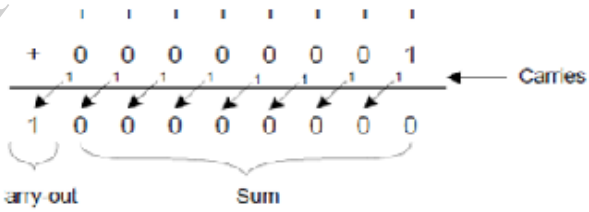

Figure 1. Binary Adder Example

The binary adder is the critical element in most digital circuit designs including digital signal processors (DSP) and microprocessor data path units. As such, extensive research continues to be focused on improving the power delay performance of the adder. 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 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 described. 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.

2. Research Contributions

The implementations that have been developed in this dissertation help to improve the design of parallel- prefix adders and their associated computing architectures. This has the potential of impacting many application specific and general purpose computer architectures. Consequently, this work can impact the designs of many computing systems, as well as impacting many areas of engineers and science. In this paper, the practical issues involved in designing and implementing tree-based adders on FPGAs are described. 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

3. Parallel-Prefix Structures

To resolve the delay of carry-look ahead adders, the scheme of multilevel-look ahead adders or parallel-prefix adders can be employed. The idea is to compute small group of intermediate prefixes and then find large group prefixes, until all the carry bits are computed. These adders have tree structures within a carry-computing stage similar to the carry propagate adder. However, the other two stages for these adders are called pre-computation and post-computation stages. In pre-computation stage, each bit computes its carry generate/propagate and a temporary sum. In the prefix stage, the group carry generate/propagate signals are computed to form the carry chain and provide the carry-in for the adder below.

$$G_{i:k} = G_{i:j} + P_{i:j} \cdot G_{j-1:k}$$
$$P_{i:k} = P_{i:j} \cdot P_{j-1:k}$$

In the post-computation stage, the sum and carry-out are finally produced. The carry-out can be omitted if only a sum needs to be produced.

$$s_i = t_i \wedge G_{i:-1}$$
$$cout = g_{n-1} + p_{n-1} \_ G_{n-2:-1}$$

where $G_{i:-1} = c_i$ with the assumption $g_{-1} = cin$.

All parallel-prefix structures can be implemented with the equations above; however, Equation can be interpreted in various ways, which leads to different types of parallel-prefix trees.

Parallel-prefix adders, also known as carry-tree adders, pre-calculate the propagate and generate signals. These signals are variously combined using the fundamental carry operator (fco) cells.

$(gL, pL) \circ (gR, pR) = (gL + pL \bullet gR, pL \bullet pR)$

Due to associative law of the *fco*, these operators can be combined in different ways to form various adder structures. For, suppose the 4-bit carry-look ahead generator is given by.

$c4 = (g4, p4) \circ [(g3, p3) \circ [(g2, p2) \circ (g1, p1)]]$

A simple rearrangement of the order of operations allows parallel operation, resulting in a more efficient tree structure for this 4-bit example

$c4 = [(g4, p4) \circ (g3, p3)] \circ [(g2, p2) \circ (g1, p1)]$

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 this study, the focus is on the Kogge-Stone adder, known for having minimal logic depth and fan-out (see Figure 3.1). Here we designate BC as the black cell which generates the ordered pair in equation; the gray cell (GC) generates the left signal only. 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.

3.1 Kogge-Stone Prefix Tree

Kogge-Stone prefix tree is among the type of prefix trees that use the fewest logic levels. A 16-bit example is shown in Figure 3.1. In fact, Kogge-Stone is a member of Knowles prefix tree. The 16-bit prefix tree can be viewed as Knowles [1, 1, 1, 1]. The numbers in the brackets represent the maximum branch fan-out at each logic level. The maximum fan-out is 2 in all logic levels for all width Kogge-Stone prefix trees.

The key of building a prefix tree is how to implement Equation according to the specific features of that type of prefix tree and apply the rules described in the previous section. 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 example in Figure 3.1 is 16-bit (a power of 2) prefix tree. It is not difficult to extend the structure to any width if the basics are strictly followed.

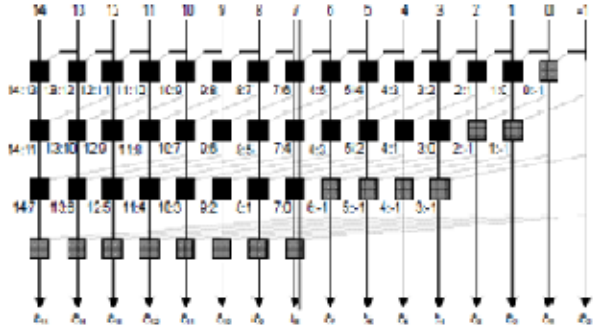

Figure 3.1: 16-bit Kogge-Stone Prefix Tree

For the Kogge-Stone prefix tree, at the logic level 1, the inputs span is 1 bit, example group (4:3) take the inputs at bit 4 and bit 3). Group (4:3) will be taken as inputs and combined with group (6:5) to generate group (6:3) at logic level 2. Group (6:3) will be taken as inputs and combined with group (10:7) to generate group (10:3) at logic level 3, and so on so forth.

Logic Levels	u	v	Output (i:i−u+1)	Input1 (i:i−v+1)	Input2 (i−v:i−u+1)	Equation Mapping
1	2	1	7:6	7:7	6:6	$GP_{7:6} = GP_7 \circ GP_6$
2	4	2	11:8	11:10	9:8	$GP_{11:8} = GP_{11:10} \circ GP_{9:8}$
3	8	4	14:7	14:11	10:7	$GP_{14:7} = GP_{14:11} \circ GP_{10:7}$
4	16	8	7:−1	7:0	−1:−1	$GP_{7:-1} = GP_{7:0} \circ GP_{-1}$

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

3.2 Sparse Kogge-Stone adder

The sparse Kogge-Stone adder, shown in Figure 3.2.This hybrid design completes the summation process with a 4 bit RCA allowing the carry prefix network to be simplified.

3.3 Spanning Tree Carry-Look ahead Adder

Another carry-tree adder is the spanning tree carry-look ahead (CLA) adder is also examined. Like the sparse Kogge-Stone adder, this design terminates with a four-bit Ripple Carry Adder (RCA). As the FPGA uses a fast carry-chain for the Ripple Carry Adder (RCA), it is interesting to compare the performance of this adder with the sparse Kogge-Stone and regular Kogge-Stone adder.

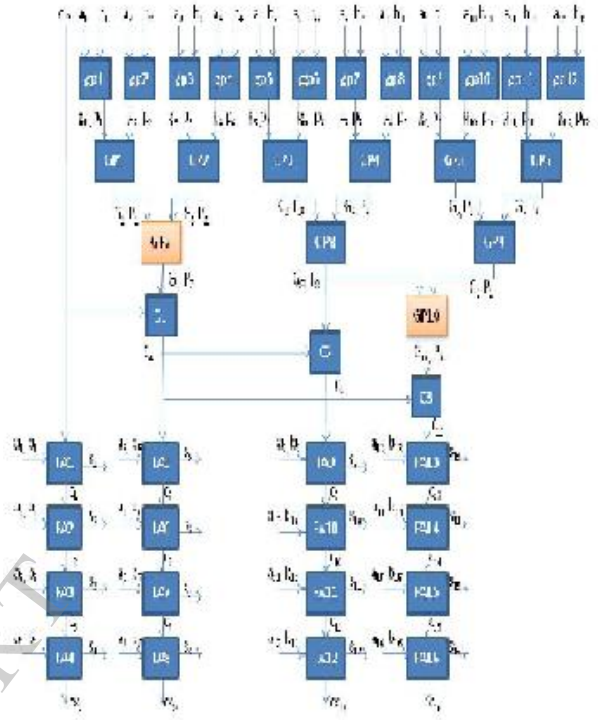

Figure 3.3: 16-bit Spanning Tree Carry Look ahead Adder.

4. Experimental Results

The Parallel Prefix adders of koggie-stone adder, sparse koggie-stone and spanning tree carry look ahead adders are simulated and synthesized verilog using Xilinx-ISE tools, and we found that, these parallel prefix adders are consuming less power and having high speed functionality as compared with the Ripple carry adder (RCA) and Carry Skip adder (CSA).The Experimental results of corresponding adders are given below.

ADDER KOGGE_STONE Partition Summary				
Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of 4 input LUTs	36	1,536	2%	
Logic Distribution				
Number of occupied Slices	24	768	3%	
Number of Slices containing only related logic	24	24	100%	
Number of Slices containing unrelated logic	0	24	0%	
Total Number of 4 input LUTs	36	1,536	2%	
Number of bonded IOBs	50	180	27%	
Total equivalent gate count for design	228			
Additional JTAG gate count for IOBs	2,400			

Figure 4.1 Design summary of Kogge-Stone Adder

The design summary of Kogge-stone adder's total number of 4-input LUTs, logic Utilization are given in 4.1 figure.

Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of 4 input LUTs	51	1,536	3%	
Logic Distribution				
Number of occupied Slices	29	768	3%	
Number of Slices containing only related logic	29	29	100%	
Number of Slices containing unrelated logic	0	29	0%	
Total Number of 4 input LUTs	51	1,536	3%	
Number of bonded IOBs	65	180	36%	
Total equivalent gate count for design	315			
Additional JTAG gate count for IOBs	3,120			

Figure 4.2 Design summary of Sparse Kogge-Stone adder.

The design summary of Sparse Kogge-stone adder's total number of 4-input LUTs, logic Utilization are given in 4.2 figure.

ADDER SPANNING_TREE Partition Summary				
Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of 4 input LUTs	32	1,536	2%	
Logic Distribution				
Number of occupied Slices	24	768	3%	
Number of Slices containing only related logic	24	24	100%	
Number of Slices containing unrelated logic	0	24	0%	
Total Number of 4 input LUTs	32	1,536	2%	
Number of bonded IOBs	65	180	36%	
Total equivalent gate count for design	192			

Figure 4.3 Design summary of Spanning Tree Carry look ahead Adder.

The design summary of Spanning tree adder's total number of 4-input LUTs, logic Utilization are given in 4.3 figure.

CARRYSKIPADDER Partition Summary				
Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of 4 input LUTs	40	1,536	2%	
Logic Distribution				
Number of occupied Slices	27	768	3%	
Number of Slices containing only related logic	27	27	100%	
Number of Slices containing unrelated logic	0	27	0%	
Total Number of 4 input LUTs	40	1,536	2%	
Number of bonded IOBs	50	180	27%	
Total equivalent gate count for design	252			
Additional JTAG gate count for IOBs	2,400			

Figure 4.4 Design summary of Carry Skip Adder.

The design summary of Carry Skip adder's total number of 4-input LUTs, logic Utilization are given in 4.4 figure.

RIPPLECARRYADDER Partition Summary				
Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of 4 input LUTs	32	1,536	2%	
Logic Distribution				
Number of occupied Slices	24	768	3%	
Number of Slices containing only related logic	24	24	100%	
Number of Slices containing unrelated logic	0	24	0%	
Total Number of 4 input LUTs	32	1,536	2%	
Number of bonded IOBs	50	180	27%	
Total equivalent gate count for design	192			
Additional JTAG gate count for IOBs	2,400			

Figure 4.5 Design summary of Ripple carry adder.

The design summary of Ripple Carry adder's total number of 4-input LUTs, logic Utilization are given in 4.5 figure.

The Experimental waveforms of Parallel Prefix adders are given below

4.6 Kogge-Stone Adder Waveform.

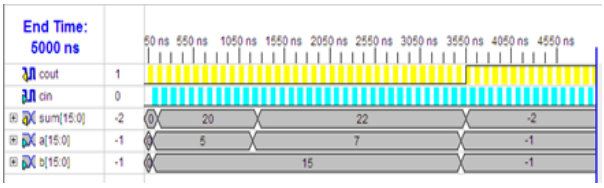

Figure 4.6 Waveform of Kogge-Stone adder

4.7 Spanning Tree Adder wave form.

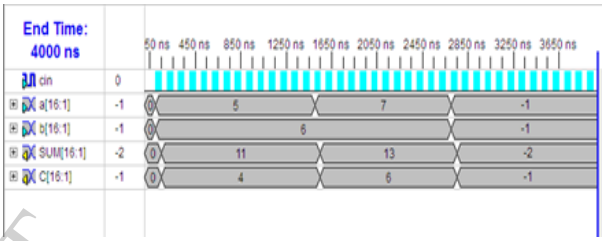

Figure 4.7 Waveform of Spanning Tree adder

4.8 Sparse Kogge-Stone Adder

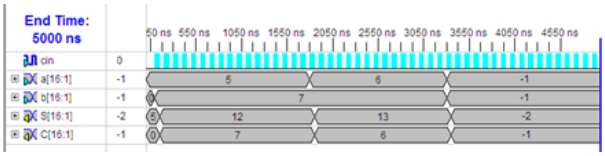

Figure 4.6 Waveform of Sparse Kogge-Stone adder

5. Conclusions

In this paper we have developed an efficient Parallel Prefix Adders to achieve less power consumption and better speed performance when compared to Ripple Carry Adder and Carry Skip Adder.

The Experimental Power Consumption Results are given below.

Ripple Carry Adder consumes 25mw.

Carry Skip Adder consumes 27mw.

Kogge-Stone Adder consumes 20mw.

Sparse Kogge-Stone Adder consumes 17mw.

Spanning Tree Adder consumes 16mw.

Based on these results, the Parallel Prefix Adders consume less power when compared to Ripple Carry

Adder and Carry Skip 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.
[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.