ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

Design and Characterization of Parallel Prefix Adders

S.Sri Mounika
Department of Electronics and Communications Engineering, St. Mary's College of Engineering & Technology, Hyderabad, Telangana-502 319, India.

K.Aksa Rani
Department of Electronics and Communications Engineering, St. Mary's College of Engineering & Technology, Hyderabad, Telangana-502 319, India.

M.S.Shyam
Department of Electronics and Communications Engineering, St. Mary's College of Engineering & Technology, Hyderabad, Telangana-502 319, India.

Abstract:
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 [1]. Parallel-prefix adders (also known as carry-tree adders) are known to have the best performance in VLSI designs. However, this performance advantage does not translate directly into FPGA implementations due to constraints on logic block configurations and routing overhead. This paper investigates three types of carry-tree adders (the Kogge-Stone, sparse Kogge-Stone, and spanning tree adder) and compares them to the simple Ripple Carry Adder (RCA) and Carry Skip Adder (CSA).

These designs of varied bit-widths were implemented on a Xilinx Spartan 3E FPGA and delay measurements were made with a high-performance logic analyzer. Due to the presence of a fast carry-chain, the RCA designs exhibit better delay performance up to 128 bits. The carry-tree adders are expected to have a speed advantage over the RCA as bit widths approach 256. In this project for simulation we use Model sim for logical verification, and further synthesizing it on Xilinx-ISE tool using target technology and performing placing & routing operation for system verification on targeted FPGA.

I. 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.

1. Binary Adders:

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 [2]. 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 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 a computer architecture, because they tend to set the critical path for most computations.

Cite this article as: S.Sri Mounika, K.Aksa Rani & M.S.Shyam, "Design and Characterization of Parallel Prefix Adders", International Journal & Magazine of Engineering, Technology, Management and Research, Volume 4, Issue 12, 2017, Page 18-31.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

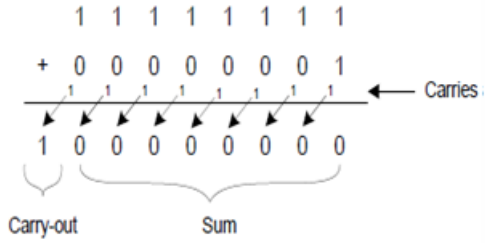

Figure 1: Binary Adder Example

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. Carry-Propagate Adders:

Binary carry-propagate adders have been extensively published, heavily attacking problems related to carry chain problem. Binary adders evolve from linear adders, which have a delay approximately proportional to the width of the adder, e.g. ripple-carry adder (RCA), to logarithmic-delay adder, such as the carry-look ahead adder (CLA) [2]. There are some additional performance enhancing schemes, including the carry-increment adder and the Ling adder that can further enhance the carry chain, however, in Very Large Scale Integration (VLSI) digital systems, the most efficient way of offering binary addition involves utilizing parallel-prefix trees, this occurs because they have the regular structures that exhibit logarithmic delay. Parallel-prefix adders compute addition in two steps: one to obtain the carry at each bit, with the next to compute the sum bit based on the carry bit [3]. Unfortunately, prefix trees are algorithmically slower than fast logarithmic adders, such as the carry propagate adders, however, their regular structures promote excellent results when compared to traditional CLA adders.

This happens within VLSI architectures because a carry-lookahead adder, such as the one implemented in one of Motorola's processors, tends to implement the carry chain in the vertical direction instead of a horizontal one, which has a tendency to increase both wire density and fan-in/out dependence. Therefore, although logarithmic adder structures are one of the fastest adders algorithmically, the speed efficiency of the carry-lookahead adder has been hampered by diminishing returns given the fan-in and 2 fan-out dependencies as well as the heavy wire load distribution in the vertical path [4]. In fact, a traditional carry-lookahead adder implemented in VLSI can actually be slower than traditional linear-based adders, such as the Manchester carry adder. The implementations that have been developed in this dissertation help to improve the design of parallel-prefix adders and their associated computing architectures.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

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.

II. RELATED STUDY:

Adders are one of the most essential components in digital building blocks, however, the performance of adders become more critical as the technology advances. The problem of addition involves algorithms in Boolean algebra and their respective circuit implementation. Algorithmically, there are linear-delay adders like ripple-carry adders (RCA), which are the most straightforward but slowest. Adders like carry-skip adders (CSKA), carry-select adders (CSEA) and carry-increment adders (CINA) are linear-based adders with optimized carry-chain and improve upon the linear chain within a ripple-carry adder. Carry-lookahead adders (CLA) have logarithmic delay and currently have evolved to parallel-prefix structures. Other schemes, like Ling adders, NAND/NOR adders and carry-save adders can help improve performance as well.

1. Binary Adder Notations and Operations:

As mentioned previously, adders in VLSI digital systems use binary notation. In that case, add is done bit by bit using Boolean equations. Consider a simple binary add with two n-bit inputs A;B and a one-bit carry-in cin along with n-bit output S.

2. Ripple-Carry Adders (RCA):

The simplest way of doing binary addition is to connect the carry-out from the previous bit to the next bit's carry-in. Each bit takes carry-in as one of the inputs and outputs sum and carry-out bit and hence the name ripple-carry adder. This type of adders is built by cascading 1-bit full adders. A 4-bit ripple-carry adder is shown in Figure 2.3. Each trapezoidal symbol represents a single-bit full adder. At the top of the figure, the carry is rippled through the adder from cin to cout.

3. Carry-Select Adders (CSEA):

Simple adders, like ripple-carry adders, are slow since the carry has to to travel through every full adder block. There is a way to improve the speed by duplicating the hardware due to the fact that the carry can only be either 0 or 1. The method is based on the conditional sum adder and extended to a carry-select adder. With two RCA, each computing the case of the one polarity of the carry-in, the sum can be obtained with a 2x1 multiplexer with the carry-in as the select signal. An example of 16-bit carry-select adder is shown in Figure 2.4. In the figure, the adder is grouped into four 4-bit blocks. The 1-bit multiplexors for sum selection can be implemented.

4. Carry-Skip Adders (CSKA):

There is an alternative way of reducing the delay in the carry-chain of a RCA by checking if a carry will propagate through to the next block. This is called carry-skip adders.

5. Carry-Look-ahead Adders (CLA):

The carry-chain can also be accelerated with carry generate/propagate logic. Carry-lookahead adders employ the carry generate/propagate in groups to generate carry for the next block [5]. In other words, digital logic is used to calculate all the carries at once. When building a CLA, a reduced version of full adder, which is called a reduced full adder (RFA) is utilized. The carry generate/propagate signals gi/pi feed to carry-lookahead generator (CLG) for carry inputs to RFA [6].

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

III. PARALLEL-PREFIX STRUCTURES:

To resolve the delay of carry-lookahead adders, the scheme of multilevel-lookahead adders or parallel-prefix adders can be employed. 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. 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.All parallel-prefix structures can be implemented with the equations; however, Equation can be interpreted in various ways, which leads to different types of parallel-prefix trees [7]. For example, Brent-Kung is known for its sparse topology at the cost of more logic levels. There are several design factors that can impact the performance of prefix structures.

- Radix/Valency
- Logic Levels
- Fan-out
- Wire tracks

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, pre-computation, prefix tree and post-computation. In the prefix tree, group generate/propagate are the only signals used. The group generate/propagate equations are based on single bit generate/propagate, which are computed in the pre-computation stage.

1. Prefix Tree Family:

Parallel-prefix trees have various architectures. These prefix trees can be distinguished by four major factors. 1) Radix/Valency 2) Logic Levels 3) Fan-out 4) Wire Tracks In the following discussion about prefix trees, the radix is assumed to be 2 (i.e. the number of inputs to the logic gates is always 2).

The more aggressive prefix schemes have logic levels [log2(n)], where n is the width of the inputs [8]. However, these schemes require higher fanout, or many wire-tracks or dense logic gates, which will compromise the performance e.g. speed or power. Some other schemes have relieved fan-out and wire tracks at the cost of more logic levels. When radix is fixed, The design trade-off is made among the logic levels, fan-out and wire tracks.

2. Preparing Prefix Tree:

The synthesis rules apply to any type of prefix tree. In this section, the methodology utilized to build fixed prefix structures is discussed. Moreover, procedure to build fixed prefix tree can be adapted to building non-fixed prefix tree with a slight modification. In general, building prefix trees can be reduced to solving the following problems.

- How to align the bit lines.
- Where to place cells that compute group generate G and propagate P, i.e. black cells in this case (gray cells are ignored here to simplify the discussion.).
- How to connect input/output of the cells

.

The solutions are based on the numbers which are power of 2 as both of the locations of the cells and wires can be related to those numbers.

To solve the problems, 3 terms are defined.

- l level: logic level;
- u: maximum output bit span;
- v: maximum input bit span;

3. Kogge-Stone Prefix Tree:

Kogge-Stone prefix tree is among the type of prefix trees that use the fewest logic levels. In fact, Kogge-Stone is a member of Knowles prefix tree. The 16-bit prefix tree can be viewed as Knowels [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.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

4. CARRY-TREE ADDER DESIGNS:

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

(gL, pL) o (gR, pR) = (gL + pL•gR, pL • pR)

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-lookahead generator is given by:

c4 = (g4, p4) o [(g3, p3) o [(g2, p2) o (g1, p1)]]

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

c4 = [(g4, p4) o (g3, p3)] o [(g2, p2) o (g1, p1)]

IV. SIMULATION RESULTS:

The corresponding simulation results of the adders are shown below.

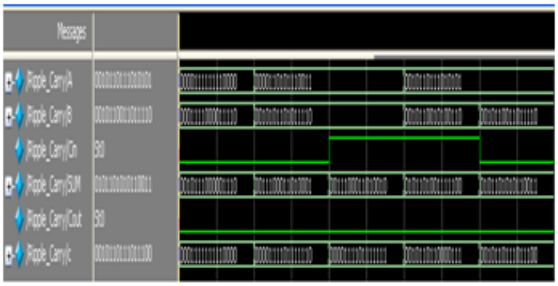

Figure 2: Ripple-Carry Adder

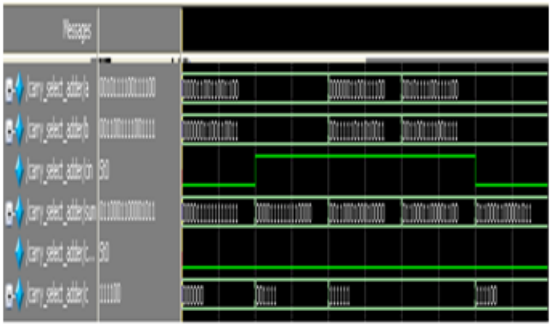

Figure 3: Carry-Select Adder

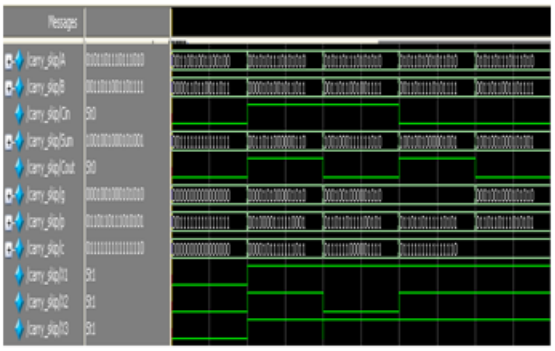

Figure 4: Carry-Skip Adder

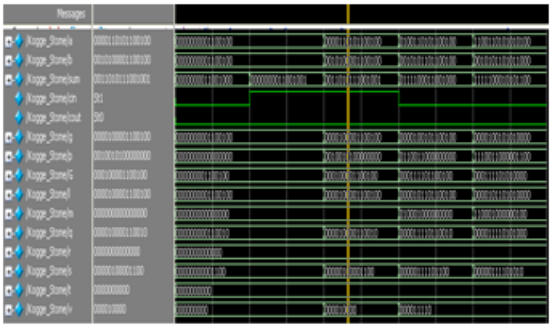

Figure 5: Kogge-Stone Adder

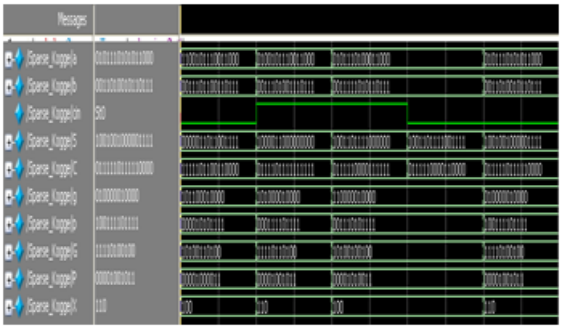

Figure 6: Sparse Kogge-Stone Adder

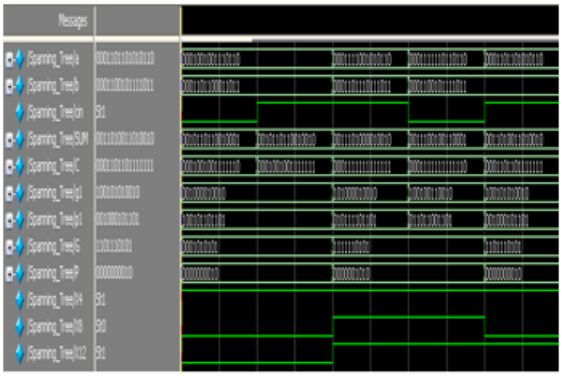

Figure 7: Spanning Tree adder

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

V. FPGA IMPLEMENTATION:

FPGA contains a two dimensional arrays of logic blocks and interconnections between logic blocks. Both the logic blocks and interconnects are programmable. Logic blocks are programmed to implement a desired function and the interconnections are programmed using the switch boxes to connect the logic blocks. Xilinx logic block consists of one Look Up Table (LUT) and one Flip-Flop. An LUT is used to implement number of different functionality. The input lines to the logic block go into the LUT and enable it. The output of the LUT gives the result of the logic function that it implements and the output of logic block is registered or unregistered output from the LUT. SRAM is used to implement a LUT.A k-input logic function is implemented using 2^k * 1 size SRAM. Number of different possible functions for k input LUT is 2^2^k. Advantage of such an architecture is that it supports implementation of so many logic functions, however the disadvantage is unusually large number of memory cells required to implement such a logic block in case number of inputs is large.

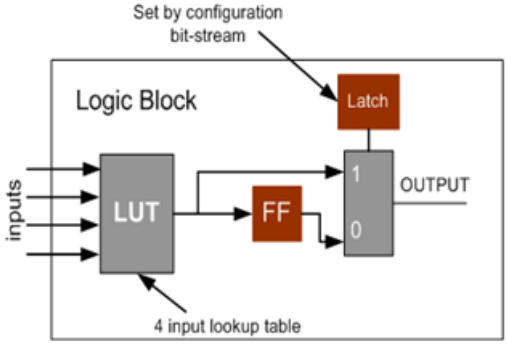

Figure 8: 4-input LUT based implementation of logic block

LUT based design provides for better logic block utilization. A k-input LUT based logic block can be implemented in number of different ways with tradeoff between performance and logic density. An n-LUT can be shown as a direct implementation of a function truth-table. Each of the latch hold's the value of the function corresponding to one input combination [9]. For Example: 2-LUT can be used to implement 16 types of functions like AND, OR, A +not B.... Etc.

In this part of tutorial we are going to have a short intro on FPGA design flow. A simplified version of design flow is given in the flowing diagram.

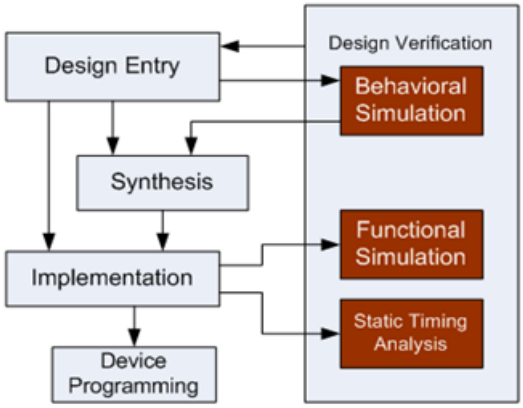

Figure 9: FPGA Design Flow

There are different techniques for design entry. Schematic based, Hardware Description Language and combination of both etc. Selection of a method depends on the design and designer. If the designer wants to deal more with Hardware, then Schematic entry is the better choice. When the design is complex or the designer thinks the design in an algorithmic way then HDL is the better choice. Language based entry is faster but lag in performance and density.

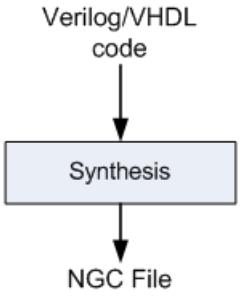

Figure 10: FPGA Synthesis

The process that translates VHDL/ Verilog code into a device netlist format i.e. a complete circuit with logical elements (gates flip flop, etc…) for the design. If the design contains more than one sub designs, ex. to implement a processor, we need a CPU as one design element and RAM as another and so on, then the synthesis process generates netlist for each design element Synthesis process will check code syntax and

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

analyze the hierarchy of the design which ensures that the design is optimized for the design architecture, the designer has selected. The resulting netlist(s) is saved to an NGC (Native Generic Circuit) file (for Xilinx® Synthesis Technology (XST)).

This process consists of a sequence of three steps

- Translate
- Map
- Place and Route

VI. SYNTHESIS RESULTS:

The corresponding schematics of the adders after synthesis are shown below.

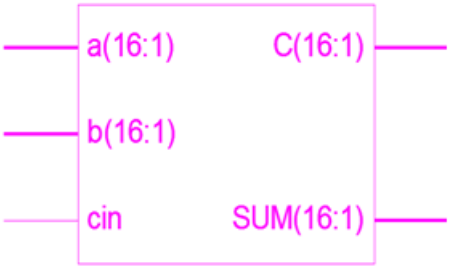

Figure 11: Top-level of Spanning Tree Adder

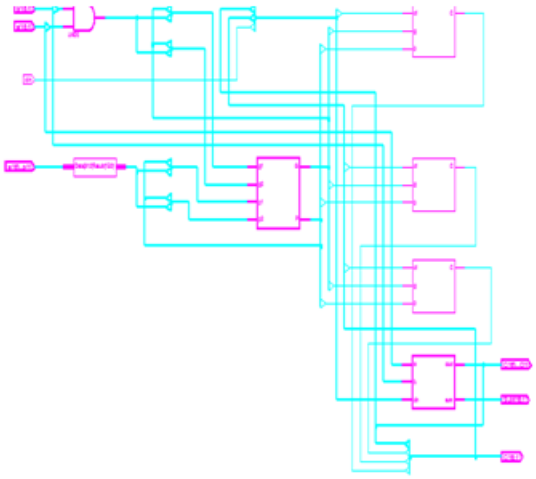

Figure 12: Internal block of Spanning Tree Adder

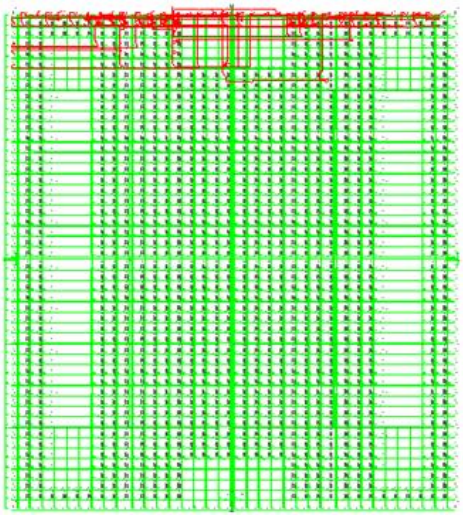

Figure 13: Area occupied by 16-bit Spanning Tree Adder

This device utilization includes the following.

- Logic Utilization
- Logic Distribution
- Total Gate count for the Design

The device utilization summery is shown above in which its gives the details of number of devices used from the available devices and also represented in %. Hence as the result of the synthesis process, the device utilization in the used device and package is shown below.

par Project Status (08/18/2011 - 12:57:42)			
Project File:	par.ise	Current State:	Placed and Routed
Module Name:	Ripple_Carry	• Errors:	No Errors
Target Device:	xc3s500e-4fg320	• Warnings:	No Warnings
Product Version:	ISE 10.1 - Foundation Simulator	• Routing Results:	All Signals Completely Routed
Design Goal:	Balanced	• Timing Constraints:	
Design Strategy:	Xilinx Default (unlocked)	• Final Timing Score:	0 (Timing Report)

par Partition Summary
No partition information was found.

Device Utilization Summary				
Logic Utilization	Used	Available	Utilization	Note(s)
Number of 4 input LUTs	32	9,312	1%	
Logic Distribution				
Number of occupied Slices	24	4,656	1%	
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	9,312	1%	
Number of bonded IOBs	50	232	21%	

Table 1: Synthesis report of Ripple-Carry Adder

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

par Project Status (08/18/2011 - 13:08:00)			
Project File:	par.ise	Current State:	Placed and Routed
Module Name:	carry_select_adder	• Errors:	No Errors
Target Device:	xc3s500e-4fg320	• Warnings:	1 Warning
Product Version:	ISE 10.1 - Foundation Simulator	• Routing Results:	All Signals Completely Routed
Design Goal:	Balanced	• Timing Constraints:	
Design Strategy:	Xilinx Default (unlocked)	• Final Timing Score:	0 (Timing Report)

par Partition Summary
No partition information was found.

Device Utilization Summary				
Logic Utilization	Used	Available	Utilization	Note(s)
Number of 4 input LUTs	47	9,312	1%	
Logic Distribution				
Number of occupied Slices	26	4,656	1%	
Number of Slices containing only related logic	26	26	100%	
Number of Slices containing unrelated logic	0	26	0%	
Total Number of 4 input LUTs	47	9,312	1%	
Number of bonded IOBs	50	232	21%	

Table 2: Synthesis report of Carry-Select Adder

par Project Status (08/18/2011 - 13:14:46)			
Project File:	par.ise	Current State:	Placed and Routed
Module Name:	carry_skip	• Errors:	No Errors
Target Device:	xc3s500e-4fg320	• Warnings:	2 Warnings
Product Version:	ISE 10.1 - Foundation Simulator	• Routing Results:	All Signals Completely Routed
Design Goal:	Balanced	• Timing Constraints:	
Design Strategy:	Xilinx Default (unlocked)	• Final Timing Score:	0 (Timing Report)

par Partition Summary
No partition information was found.

Device Utilization Summary				
Logic Utilization	Used	Available	Utilization	Note(s)
Number of 4 input LUTs	40	9,312	1%	
Logic Distribution				
Number of occupied Slices	25	4,656	1%	
Number of Slices containing only related logic	25	25	100%	
Number of Slices containing unrelated logic	0	25	0%	
Total Number of 4 input LUTs	40	9,312	1%	
Number of bonded IOBs	50	232	21%	

Table 3: Synthesis report of Carry-Skip Adder

par Project Status (08/18/2011 - 12:42:13)			
Project File:	par.ise	Current State:	Placed and Routed
Module Name:	Kogge_Stone	• Errors:	No Errors
Target Device:	xc3s500e-4fg320	• Warnings:	4 Warnings
Product Version:	ISE 10.1 - Foundation Simulator	• Routing Results:	All Signals Completely Routed
Design Goal:	Balanced	• Timing Constraints:	
Design Strategy:	Xilinx Default (unlocked)	• Final Timing Score:	0 (Timing Report)

par Partition Summary
No partition information was found.

Device Utilization Summary				
Logic Utilization	Used	Available	Utilization	Note(s)
Number of 4 input LUTs	37	9,312	1%	
Logic Distribution				
Number of occupied Slices	24	4,656	1%	
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	37	9,312	1%	
Number of bonded IOBs	50	232	21%	

Table 4: Synthesis report of Kogge-Stone Adder

par Project Status (08/18/2011 - 12:33:15)			
Project File:	par.ise	Current State:	Placed and Routed
Module Name:	Sparse_Kogge	• Errors:	No Errors
Target Device:	xc3s500e-4fg320	• Warnings:	No Warnings
Product Version:	ISE 10.1 - Foundation Simulator	• Routing Results:	All Signals Completely Routed
Design Goal:	Balanced	• Timing Constraints:	
Design Strategy:	Xilinx Default (unlocked)	• Final Timing Score:	0 (Timing Report)

par Partition Summary
No partition information was found.

Device Utilization Summary				
Logic Utilization	Used	Available	Utilization	Note(s)
Number of 4 input LUTs	51	9,312	1%	
Logic Distribution				
Number of occupied Slices	30	4,656	1%	
Number of Slices containing only related logic	30	30	100%	
Number of Slices containing unrelated logic	0	30	0%	
Total Number of 4 input LUTs	51	9,312	1%	
Number of bonded IOBs	65	232	28%	

Table 5: Synthesis report of Sparse Kogge-Stone Adder

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

par Project Status (08/18/2011 - 12:45:20)			
Project File:	par.ise	Current State:	Placed and Routed
Module Name:	Spanning_Tree	• Errors:	No Errors
Target Device:	xc3s500e-4fg320	• Warnings:	3 Warnings
Product Version:	ISE 10.1 - Foundation Simulator	• Routing Results:	All Signals Completely Routed
Design Goal:	Balanced	• Timing Constraints:	
Design Strategy:	Xilinx Default (unlocked)	• Final Timing Score:	0 (Timing Report)

par Partition Summary
No partition information was found.

Device Utilization Summary				
Logic Utilization	Used	Available	Utilization	Note(s)
Number of 4 input LUTs	32	9,312	1%	
Logic Distribution				
Number of occupied Slices	24	4,656	1%	
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	9,312	1%	
Number of bonded IOBs	65	232	28%	

Table 6: Synthesis report of Spanning Tree Adder

VII. CONCLUSION AND FUTURE WORK:

Both measured and simulation results from this study have shown that parallel-prefix adders are not as effective as the simple ripple-carry adder at low to moderate bit widths. This is not unexpected as the Xilinx FPGA has a fast carry chain which optimizes the performance of the ripple carry adder. However, contrary to other studies, 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 crypto graphical 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. This would improve their performance similar to what is found for the RCA.

We plan to explore possible FPGA architectures that could implement a "fast-tree chain" and investigate the possible trade-offs involved. The built-in redundancy of the Kogge-Stone carry-tree structure and its implications for fault tolerance in FPGA designs is being studied. The testability and possible fault tolerant features of the spanning tree adder are also topics for future research

REFERENCES:

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

[2] 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.

[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] 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.

[6] T. Lynch and E. E. Swartzlander, "A Spanning Tree Carry Look ahead Adder," IEEE Trans. on Computers, vol. 41, no. 8, pp. 931-939, Aug. 1992.

[7] 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.

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

[9] 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. 172.