International Journal of Innovative Research in Electronics and Communications (IJIREC)
Volume 2, Issue 5, July 2015, PP 8-16
ISSN 2349-4042 (Print) & ISSN 2349-4050 (Online)
www.arcjournals.org

Speed Comparison Parallel Prefix Adders with RCA and CSA in FPGA

Dr. A Maheswara Reddy[1], G V Vijayalakshmi[2]

[1]Professor, ECE Department, PBR VITS, Kavali.
amreddy73@gmail.com
[2]PG Student, PBR VITS, Kavali.
gvvlakshmi@gmail.com

Abstract: *The Adders are the critical elements in most of the digital circuit designs, including digital signal processors (DSP) and microprocessors. Extensive research has gone into the VLSI implementations of Parallel Prefix Adders which are known for their best performance.*

The performance of Parallel Prefix Adders is directly affected by the constraints in the logic implementations of Parallel Prefix Adders in FPGA. This paper investigates the delay performance of three types of Carry Tree Adders (the Kogge-Stone, sparse Kogge-Stone, and spanning tree adder) and compares them with 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.

This comparison study is done using Modelsim for logical verification. Synthesizing was done using Xilinx-ISE tool.

Keywords: *Adders, Kogge Stone, Ripple Carry, Carry Select.*

1. INTRODUCTION

DIGITAL computer arithmetic is an aspect of logic design with the objective of developing appropriate algorithms in order to achieve an efficient utilization of the available hardware [11-14].Given that the hardware can only perform a relatively simple and primitive set of Boolean operations, arithmetic operations are based on a hierarchy of operations that are built upon the simple ones. Since ultimately, speed, power and chip area are the most often used measures of the efficiency of an algorithm, there is a strong link between the algorithms and technology used for its implementation

2. HIGH-SPEED ADDITION

Algorithms and VLSI Implementation First we will examine a realization of a one-bit adder which represents a basic building block for all the more elaborate addition schemes.

1) Full Adder

Operation of a Full Adder is defined by the Boolean equations for the sum and carry signals

$$si = aibici + aibici + aibici + aibici$$

$$= ai \oplus bi \oplus ci$$

$$ci+ = aibici + aibici + aibici + aibici$$

Where ai, bi, and ci are the inputs to the i − th full adder stage, and si and ci+1 are the sum and carry outputs from the i − th stage, respectively. From the above equation we realize that the realization of the Sum function requires two XOR logic gates. Propagate pi and Carry-Generate gi terms

$$si = ai \oplus bi, ci = ai \cdot bi$$

At a given stage i, a carry is generated if gi is true (i.e., both ai and bi are ONEs), and if pi is true a

stage propagates an input carry to its output (i.e., either ai or bi is a ONE). The logical implementation of the full adder is shown in Figure 1.

For this implementation, the delay from either ai or bi to si is two XOR delays and the delay from ci to ci+1 is 2 gate delays. Some technologies, such as CMOS, implement the functions more efficiently

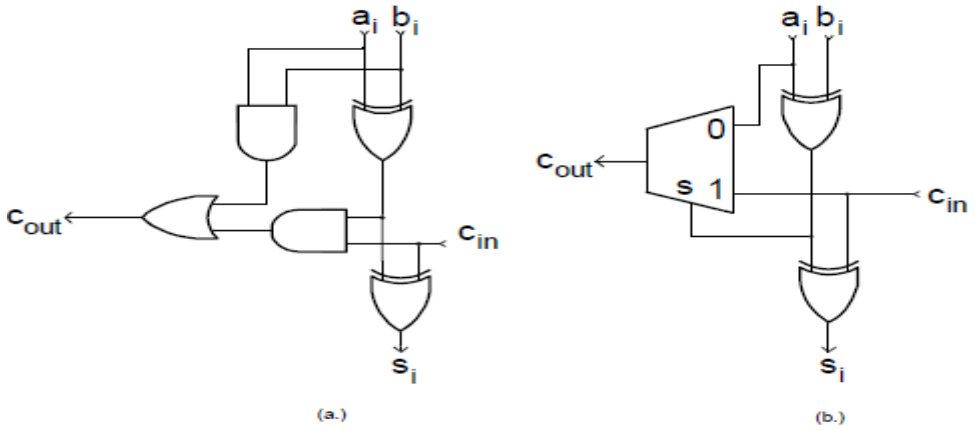

Fig. 1. *Full-Adder implementation (a) regular (b) using multiplexer in the critical path*

by using pass-transistor circuits. For example, the critical path of the carry-in to carry-out uses a fast pass-transistor multiplexer [15] in an alternative implementation of the Full Adder shown in Fig.1.b.The ability of pass-transistor logic to provide an efficient multiplexer implementation has been exploited in CPL and DPL logic families [10,11]. Even an XOR gate is more efficiently implemented using multiplexer topology. A Full-Adder cell which is entirely multiplexer based as published by Hitachi [11] is shown in Fig.2. Such a Full-Adder realization contains only two transistors in the Input-to-Sum path and only one transistor in the Cin-to-Cout path (not counting the buffer). The short critical path is a factor that contributes to a remarkable speed of this implementation.

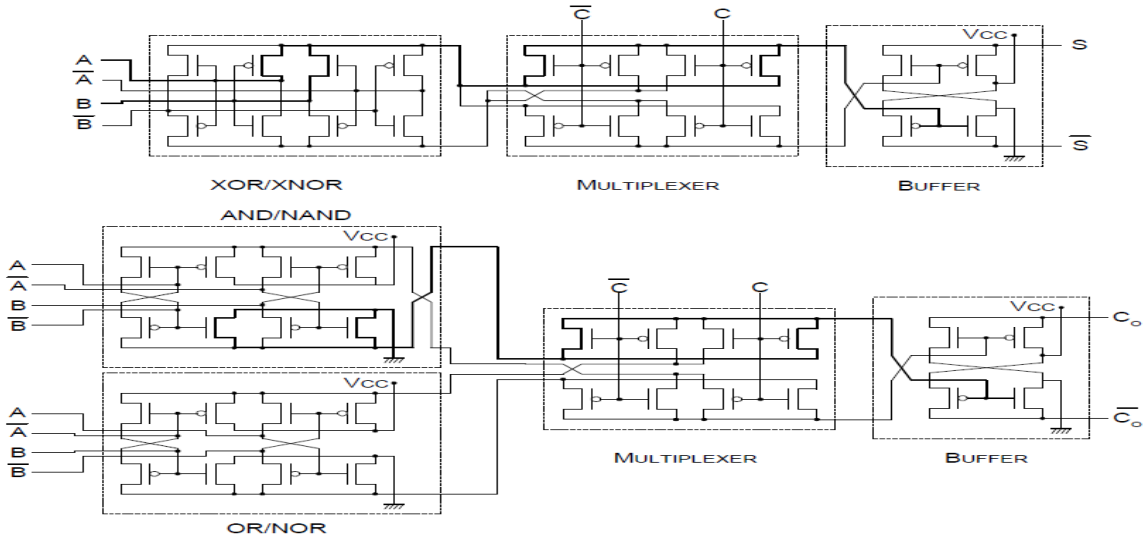

Fig. 2. *Pass-Transistor realization of a Full-Adder in DPL [11]*

2) Ripple Carry Adder

A ripple carry adder for N-bit numbers is implemented by concatenating N full adders as shown on Figure3. At the i-th bit position, the i-th bits of operands A and B and a carry signal from the preceding adder stage are used to generate the i th bit of the sum, si, and a carry,ci+1, to the next adder stage. This is called a Ripple Carry Adder (RCA), since the carry signal ripple from the least significant bit position to the most significant [1314]. If the ripple carry adder is implemented by concatenating N full adders, the delay of such an adder is 2N gate delays from Cin-to-Cout. The path from the input to the output signal that is likely to take the longest time is designated as a ”critical path”. In the case of a RCA, this is the path from the least significant input a0 or b0 to the last sum

bit sn. Assuming a multiplexer based XOR gate implementation, this critical path will consist of N+1 pass transistor delays. However, such a long chain of transistors will significantly degrade the signal, thus some application points are necessary. In practice, we can use a multiplexer cell to build this critical path using standard cell library as shown in Fig3

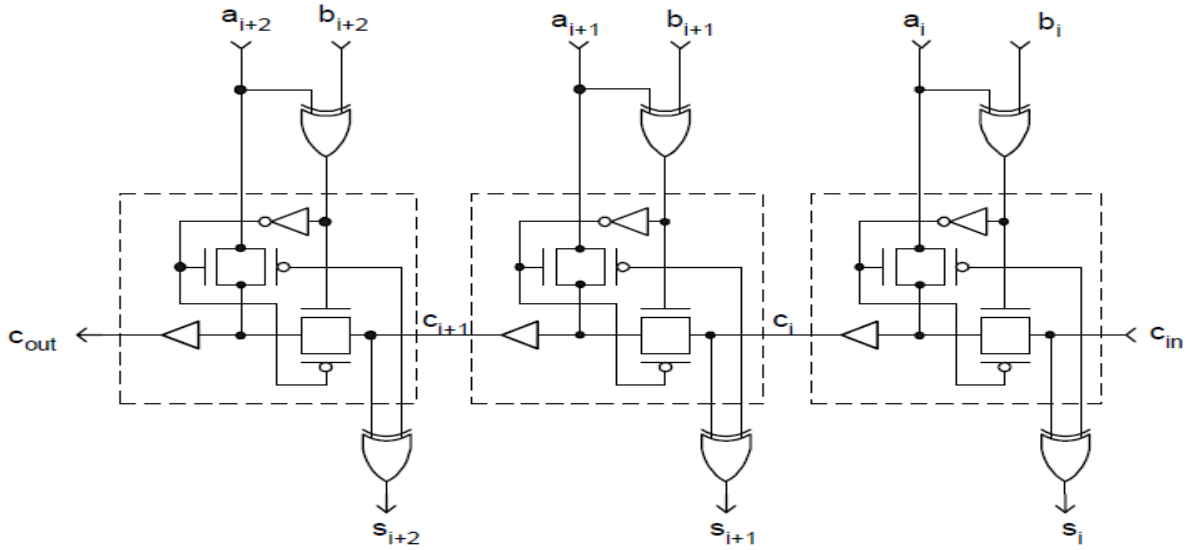

Fig. 3. *Carry-Chain of an RCA implemented using multiplexer from the standard cell library [8]*

3) Carry Skip Adder

Since the Cin -to -Cout represents the longest path in the ripple-carry-adder an obvious attempt is to accelerate carry propagation through the adder. This is accomplished by using Carry-Propagate pi signals within a group of bits. If all the pi signals within the group are pi =1, the condition exist for the carry to bypass the entire group

$$p = pi.pi+1.pi+2 \ldots pi+k \quad (1)$$

The Carry Skip Adder (CSKA) divides the words to be added into groups of equal size of k-bits. The basic structure of an N-bit Carry Skip Adder is shown on Fig.4. Within the group, carry propagates in a ripple-carry fashion. In addition, an AND gate is used to form the group propagate signal P. If P = 1 the condition exists for carry to bypass (skip) over the group as shown in Fig4.

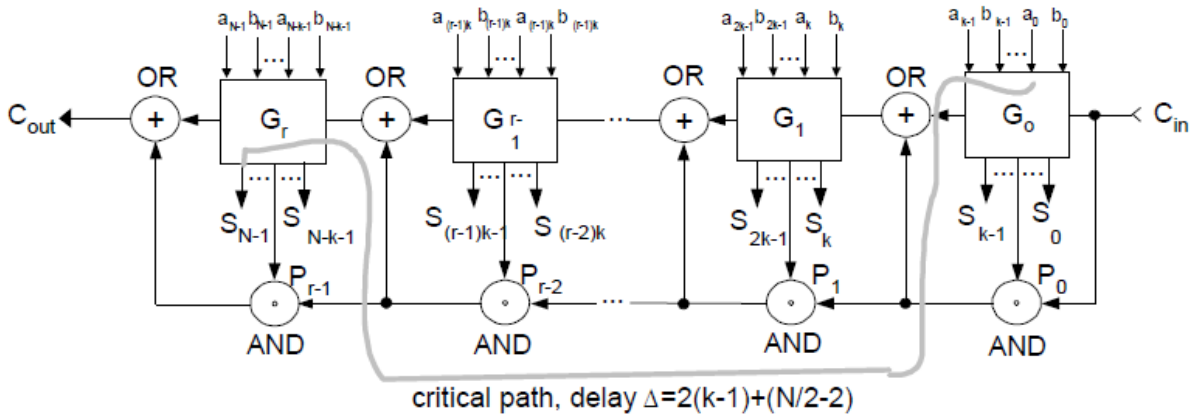

Fig. 4. *Basic Structure of a CSA: N-bits, k-bits/group, r = N/k groups*

The maximal delay of a Carry Skip Adder is encountered when carry signal is generated in the least-significant bit position, rippling through k − 1 bit positions, skipping over N/k − 2 groups in the middle, rippling to the k −1 bits of most significant group and being assimilated in the Nth bit position to produce the sum SN N +(Although it would be possible to continue this

$$\Delta CSA = (K - 1)\Delta rca - 2)\Delta skip + (K - 1)\Delta rca\ 2\ N = 2(K - 1)\Delta rca + (-2)\Delta skip$$

Thus, CSA is faster than RCA at the expense of a few relatively simple modifications. The delay is still linearly dependent on the size of the adder N, however this linear dependence is reduced by a factor of 1/k[3].

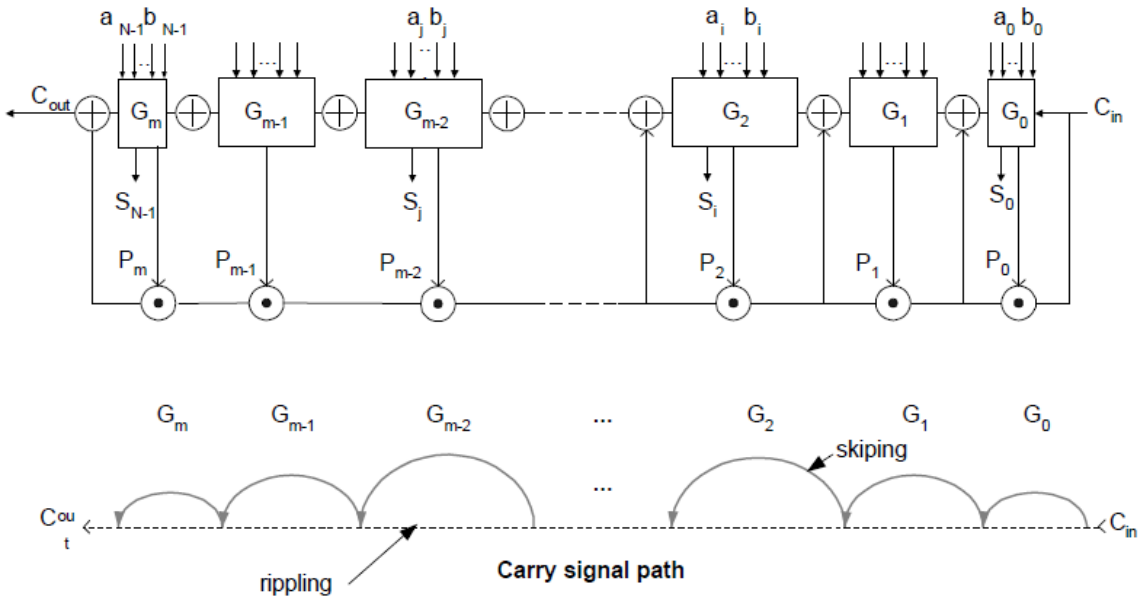

Fig. 5. *Carry-chain of a 32-bit Variable Block Adder*

4) Carry Look Ahead Adder

A significant speed improvement in the implementation of a parallel adder was introduced by a Carry-Lookahead-Adder (CLA) developed by Weinberger and Smith in 1958 [13]. The CLA adder is theoretically one of the fastest schemes used for the addition of two numbers, since the delay to add two numbers depends on the logarithm of the size of the operands Δ= logINl

The Carry Loookahead Adder uses modified full adders (modified in the sense that a carry output is not formed) for each bit position and Lookahead modules which are used to generate carry signals in-dependently for a group of k-bits. In most common case k =4. In addition to carry signal for the group, Lookahead modules produce group carry generate G and group carry propagate P outputs that indicate that a carry is generated within the group, or that an incoming carry would propagate across the group.c Extending the carry equation to a second stage in a Ripple-Carry-Adder we obtain:

$$c_{i+2} = g_{i+1} + p_{i+1}c_{i+1}$$
$$= g_{i+1} + p_{i+1}(g_i + p_i c_1)$$
$$= g_{i+1} + p_{i+1}g_i + p_{i+1}p_i c_1$$

process indefinitely, each additional stage increases the size (i.e., the number of inputs) of the logic gates. Three inputs (as required to implement ci+2 equation) is frequently the maximum number of inputs per gate for current technologies. To continue the process, Carry-Lookahead utilizes group generate and group propagate signals over three bit groups (stages i to i +2), Gi and Pi, respectively:

$$G_i = g_{i+2} + p_{i+2}g_{i+1} + p_{i+2}p_{i+1}g_i$$

$$P_i = p_{i+2}.p_{i+1}.p_i$$

$$c_{i+1} = G_i + P_i c_i$$

3. Carry Tree Adders

A. Introduction

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).

$$(g_L, p_L) \bullet (g_R, p_R) = (g_L + p_L.g_R, p_L.p_R) \quad (2)$$

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) • [(g3,p3) • [(g2,p2) • (g1,p1)]] (3)

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) • (g3,p3)] • [(g2,p2) • (g1,p1)]] (4)

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 fanout (see Figure 6). Here we designate BC as the black cell which generates the ordered pair in equation 2; 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. The sparse Kogge-Stone adder, shown in Figure 7, is also studied. This hybrid design completes the summation process with a 4 bit RCA allowing the carry prefix network to be simplified.

Fig. 6. *16 bit Kogge-Stone adder*

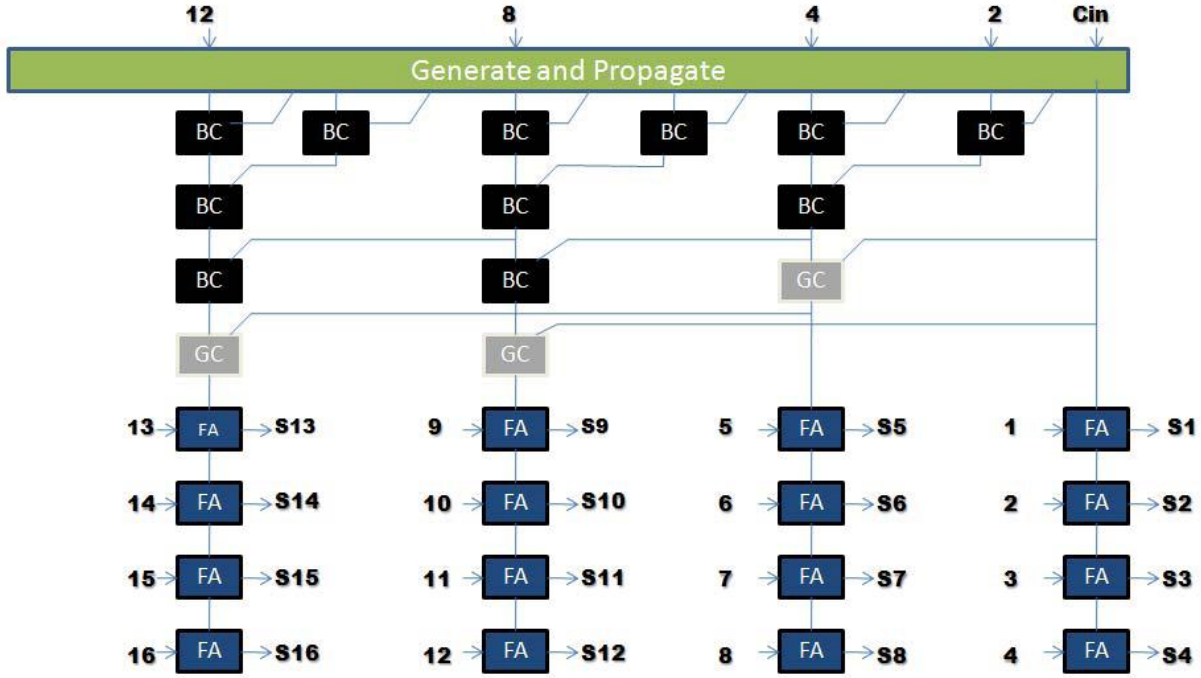

Fig. 7. *Sparse 16 bit Kogge-Stone adder*

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 features [7].

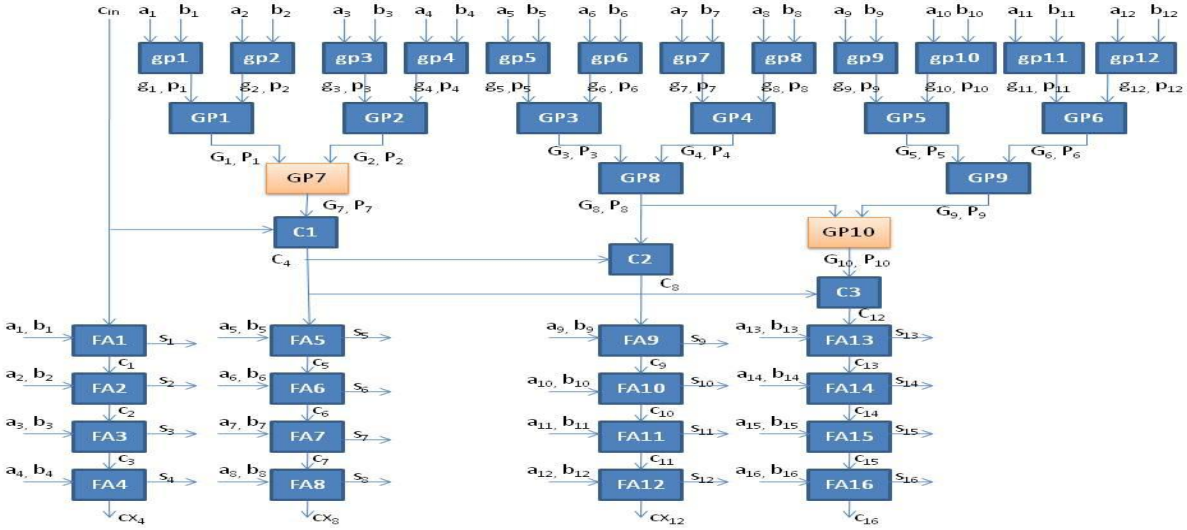

Fig. 8. *16-bit Spanning Tree Carry Lookahead Adder*

4. 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 ModelSim. The Xilinx ISE 12.2 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 (labeled as ROM in the figure below) was instantiated on the FPGA using the CoreGenerator 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. 3. 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 multiplexers,and interconnect (both external cabling and internal routing).

Second, the parallel prefix network was analyzed to determine if a specific pattern could be used to extract the worst case delay. Considering the structure of the Generate Propagate (GP) blocks

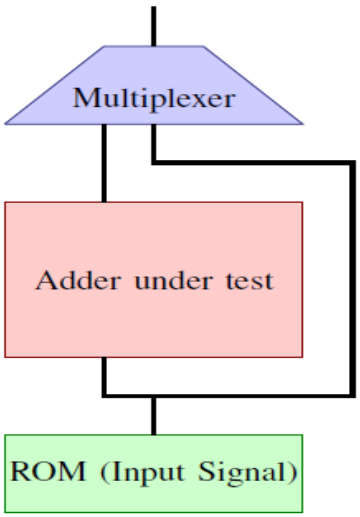

Fig. 9. Circuit used to test the adders

(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. 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.

5. DISCUSSION OF RESULTS

The simulated adder delays obtained from the Xilinx ISE synthesis reports are shown in Fig. 10. The simulation results for the carry skip adders are not included because the ISE software is not able to correctly identify the critical path through the adder and hence does not report accurate estimates of the adder delay. Observe that a semi-log plot is employed, so as expected the tree-adder delay plots as a straight line on this graph. Somewhat surprising is the fact that the sparse Kogge-Stone adder has about the same delay as the regular Kogge-Stone adder.

Table I. *Subset of (G,P) Relations Used for Testing*

(gl, pl)(gR, pR)	(gL + pLgR, pLpR)
(0,1) (0,1)	(0,1)
(0,1) (1,0)	(1,0)
(0,1) (0,1)	(1,0)
(1,0) (1,0)	(1,0)

Because the sparse Kogge-Stone completes the summation process with a 4 bit RCA, which are optimized via the fast carry chain, its performance is expected to be intermediate between the regular Kogge-Stone adder and the RCA. The impact of the routing overhead would seem to be a likely cause. However, according to the synthesis reports, the delay with the logic only makes the regular KoggeStone slightly faster. This will need to be a topic of further investigation.

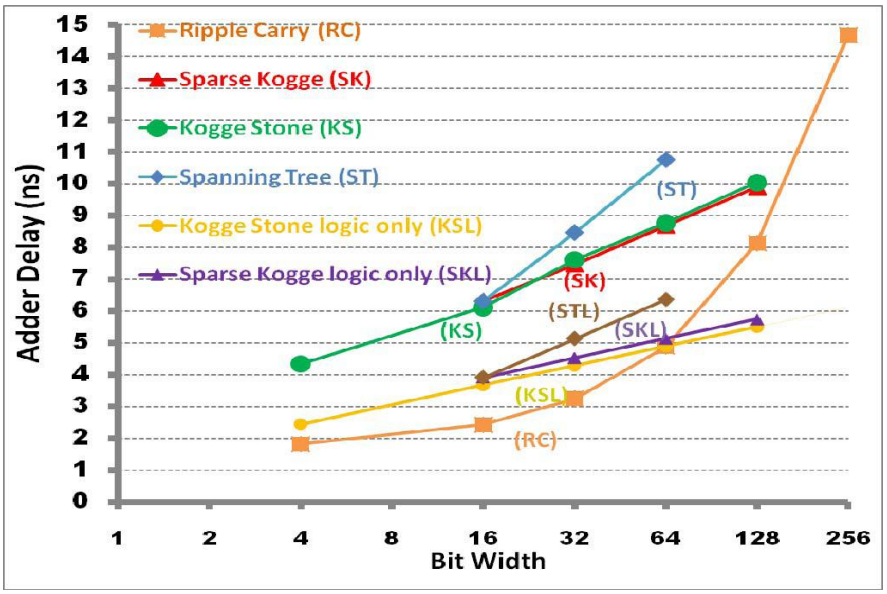

Fig. 10. *Simulation results for the adder designs.*

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:

$$tKS = (n + 2)\Delta LUT + \rho KS(n) \quad (5)$$

Table 2. *Subset of (G<P) Relations used for testing.*

N	Synth Predict	Route Delay	Route Fitted	Delay t_{KS}	Delay t_{RCA}
4	4.343	1.895	1.852	4.3	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	3.8	8.696	4.877
128	10.038	4.53	4.552	10.06	8.141
256			5.41	11.53	14.669

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:

$$tRCA = (N2).\Delta MUX + \tau RCA \quad (6)$$

where ΔMUX is the mux delay associated with the fast-carry chain and τRCA 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 =0.612ns, ΔMUX =0.051ns, and τRCA =1.715ns. 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 (N2).ΔMUX . Indeed, Table V predicts that the Kogge-Stone adder will have superior performance at N = 256

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 Virtex 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

6. SUMMARY

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 cryptographical 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, PearsonAddison-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 Asilomar Conf. Signals Systems and Computers, pp. 22137, 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. Bev and P. tukjunger, 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. 172”

[11]. ”A. Avizienis, Digital Computer Arithmetic: A Unified Algorithmic Specificatioin, Symposium on Computers and Automata, Polytechnic Institute of Brooklyn, April 13-15, 1971”.

[12]. ”Earl E. Swartzlander, Computer Arithmetic Vol. 1&2, IEEE Com puter Society Press,1990”.

[13]. K. Hwang, ”Computer Arithmetic : Principles, Architecture and Design”, John Wiley andSons, 1979

[14]. ”S.Waser, M.Flynn, Introduction to Arithmetic for Digital Systems Diesigners, Holt, Rinehart and Winston 1982”