Fall 11-7-2013

FPGA fault tolerant arithmetic logic: A case study using parallel-prefix adders

David H.K. Hoe
The University of Texas at Tyler, dhoe@uttyler.edu

Deepthi Bollepalli
The University of Texas at Tyler

Chris D. Martinez
The University of Texas at Tyler

Hindawi Publishing Corporation
VLSI Design
Volume 2013, Article ID 382682, 10 pages
http://dx.doi.org/10.1155/2013/382682

Research Article

FPGA Fault Tolerant Arithmetic Logic: A Case Study Using Parallel-Prefix Adders

David H. K. Hoe, L. P. Deepthi Bollepalli, and Chris D. Martinez

Department of Electrical Engineering, The University of Texas at Tyler, Tyler, TX 75799, USA

Correspondence should be addressed to David H. K. Hoe; dhoe@uttyler.edu

Received 5 August 2013; Accepted 19 September 2013

Academic Editor: Chien-Min Ou

This paper examines fault tolerant adder designs implemented on FPGAs which are inspired by the methods of modular redundancy, roving, and gradual degradation. A parallel-prefix adder based upon the Kogge-Stone configuration is compared with the simple ripple carry adder (RCA) design. The Kogge-Stone design utilizes a sparse carry tree complemented by several smaller RCAs. Additional RCAs are inserted into the design to allow fault tolerance to be achieved using the established methods of roving and gradual degradation. A triple modular redundant ripple carry adder (TMR-RCA) is used as a point of reference. Simulation and experimental measurements on a Xilinx Spartan 3E FPGA platform are carried out. The TMR-RCA is found to have the best delay performance and most efficient resource utilization for an FPGA fault-tolerant implementation due to the simplicity of the approach and the use of the fast-carry chain. However, the superior performance of the carry-tree adder over an RCA in a VLSI implementation makes this proposed approach attractive for ASIC designs.

1. Introduction

Field programmable gate arrays (FPGAs) are growing in popularity with designers as their ability to be reconfigured from a desktop computer allows ease of prototyping, rapid time to market, and minimal nonrecurring engineering (NRE) cost compared to custom integrated circuit (IC) designs. As FPGAs can be configured precisely for a given function, they often offer superior performance in terms of speed and power compared with using a general purpose microprocessor-based design. Hence, FPGA-based designs can form an integral part of a high-performance computing platform. FPGA manufacturers are using state-of-the-art IC processes in order to improve performance. For example, the Virtex 7 FPGA from Xilinx is implemented in a process with 28 nm feature sizes. Currently, leading edge IC designs are at the 22 nm technology node and by next year will be at the 14 nm node. With these nanometric feature sizes, ICs become more susceptible to faults from external sources like electromagnetic interference and cosmic radiation, leading to single-event upsets (SEUs) and multibit upsets (MBUs). As such, fault tolerant design is important for mission-critical applications, such as avionic and medical electronics, and for the improved reliability for space-based systems operating in harsh and remote environments. A fault tolerant system has two main features: (1) the ability to detect faults, and (2) the means to recover from the fault. An ideal fault tolerant system will perform these functions with a minimum amount of temporal and area overhead while having the ability to handle a variety of fault conditions.

This paper focuses on methods for implementing fault-tolerant arithmetic logic on FPGAs. The adder is often the critical component in the datapath of an embedded processor or in the multiply-accumulate unit (MAC) in a digital signal processing block. As such, the methods discussed in this paper illustrate techniques for implementing fault tolerance on a critical component that could be applied to other systems on an FPGA. This paper expands upon our work previously reported in conference papers regarding the characterization of arithmetic logic on FPGAs [2] and the implementation of fault tolerant adder designs on FPGAs [3]. In particular, this paper examines the implementation of fault tolerance in adders that utilize a parallel-prefix scheme for rapid computation of the carry signals. Prior work in this area has attempted to exploit the inherent redundancy within the carry tree of the Kogge-Stone adder [4, 5]. Our approach is

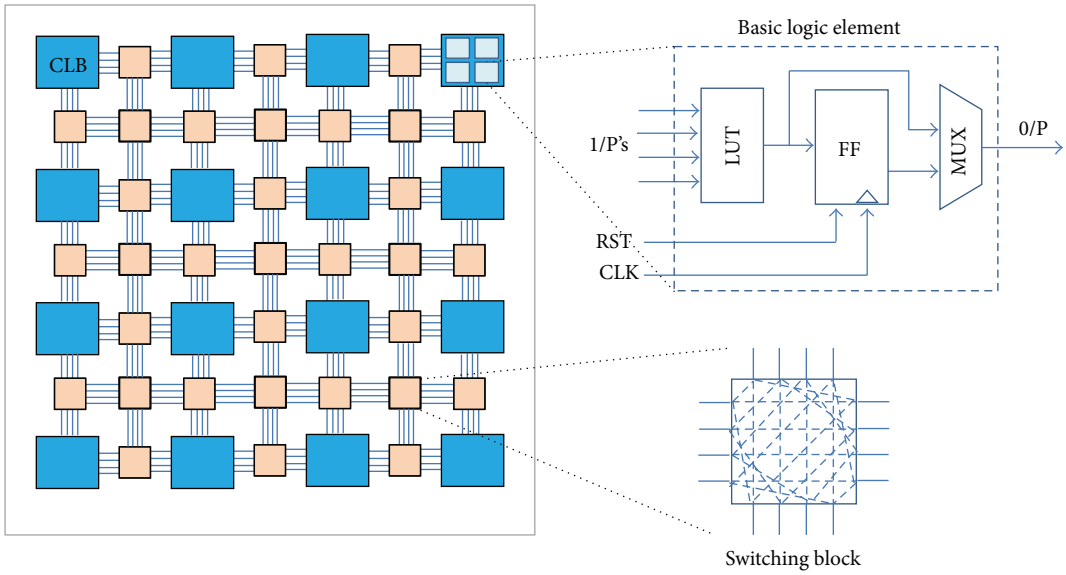

Figure 1: Structure of the typical SRAM-based FPGA.

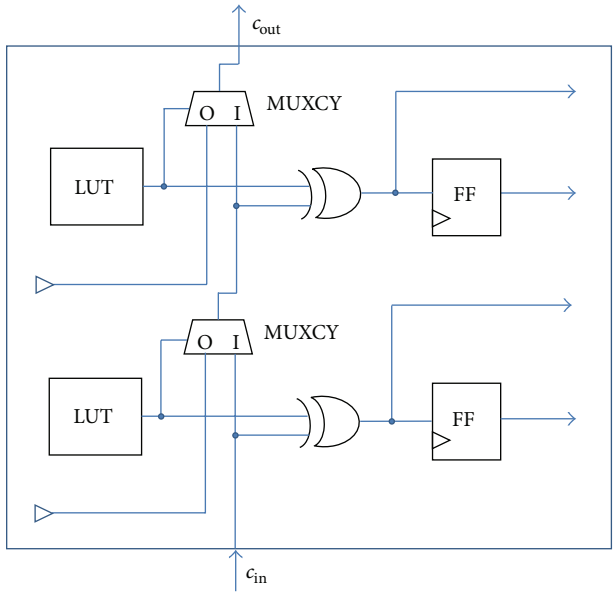

Figure 2: Illustration of the fast carry chain for one slice consisting of two BLEs in a Xilinx FPGA [1].

to implement a Kogge-Stone adder with a sparse carry tree combined with several smaller ripple carry adders (RCAs). This allows the fault tolerant methods of roving and gradual degradation to be explored. The RCA implemented with the standard method of Triple Mode Redundancy (TMR) is used as a point of reference. This paper is outlined as follows. The second section provides background information regarding the structure of the modern FPGA, compares the implementation of RCAs on FPGAs with parallel-prefix designs, and reviews relevant prior works on implementing fault tolerant systems on FPGAs. This information is key for understanding the fault tolerant designs based upon the Sparse Kogge-Stone (SKS) adder presented in the third section. The fourth section presents experimental results from this investigation. The final section provides some conclusions and discussion of future work in this area.

2. Background and Previous Work

This section provides an overview of fault tolerant techniques that are relevant for our discussion on FPGA designs. We begin by reviewing the structure of the FPGA and note the use of circuits designed to accelerate arithmetic operations, most notably the fast carry chain. The design of parallel-prefix adders and their performance relative to the simple RCA are reported. This is important for understanding our fault tolerant approach for wide-bit adders discussed in the next section. Some important fault tolerant techniques that are relevant for our discussion are also overviewed in this section.

2.1. SRAM-Based FPGA Architecture. The island-like structure that generally typifies the modern SRAM-based FPGA is illustrated in Figure 1. Configurable Logic Blocks (CLBs) are arranged in a matrix-like grid surrounded by programmable interconnect. Most modern FPGAs use a hierarchical architecture where each CLB will consist of a cluster of basic logic elements (BLEs). Each BLE is composed of a lookup-table (LUT) capable of implementing arbitrary logic functions and a flip-flop for storing the output of the LUT. This provides the option of creating one element of a state machine in each BLE. A CLB will typically consist of several BLEs. A routing hierarchy exists where more efficient routing at a local level can occur between BLEs compared with routing between CLBs at intermediate and global levels. This simplifies the complexity of the programmable interconnect that exists between CLBs, which includes a series of switching boxes. Not shown are the specialized functional blocks, such as DSP blocks and block RAM, that are often included to boost performance for many

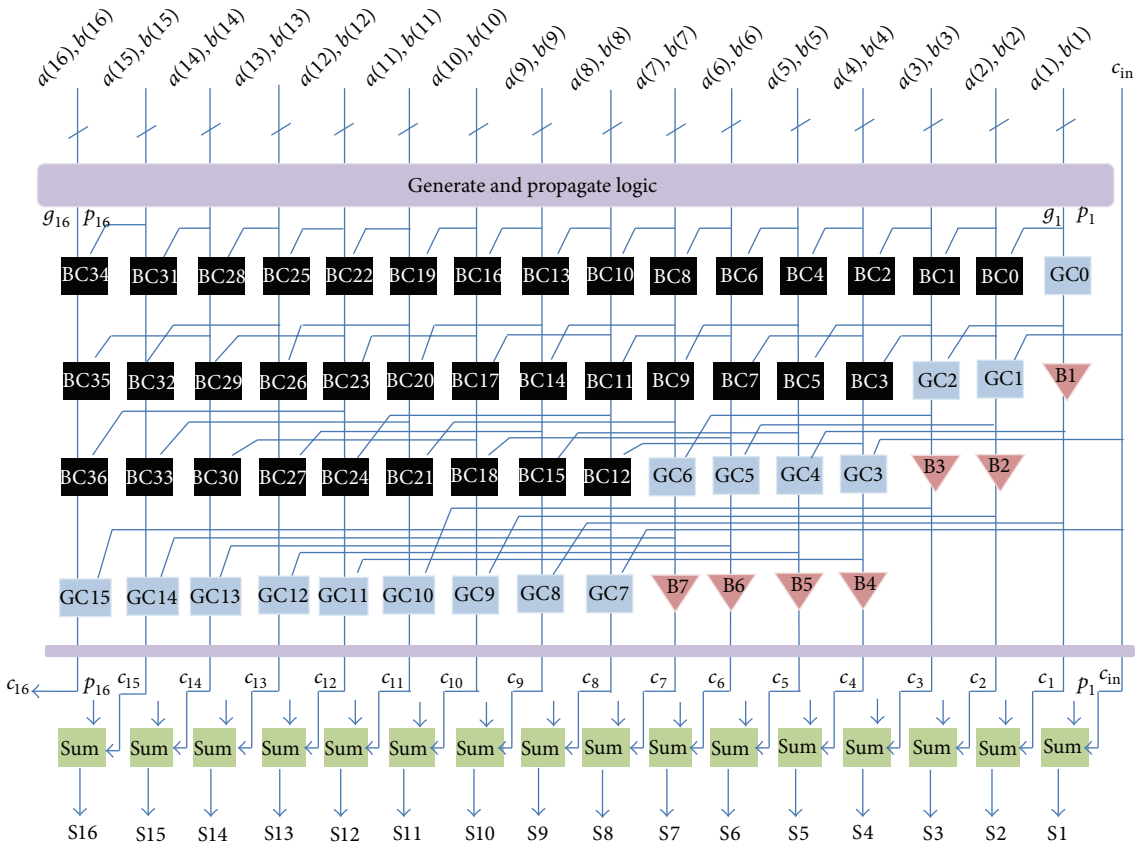

Figure 3: Illustration of a 16-bit Kogge-Stone adder.

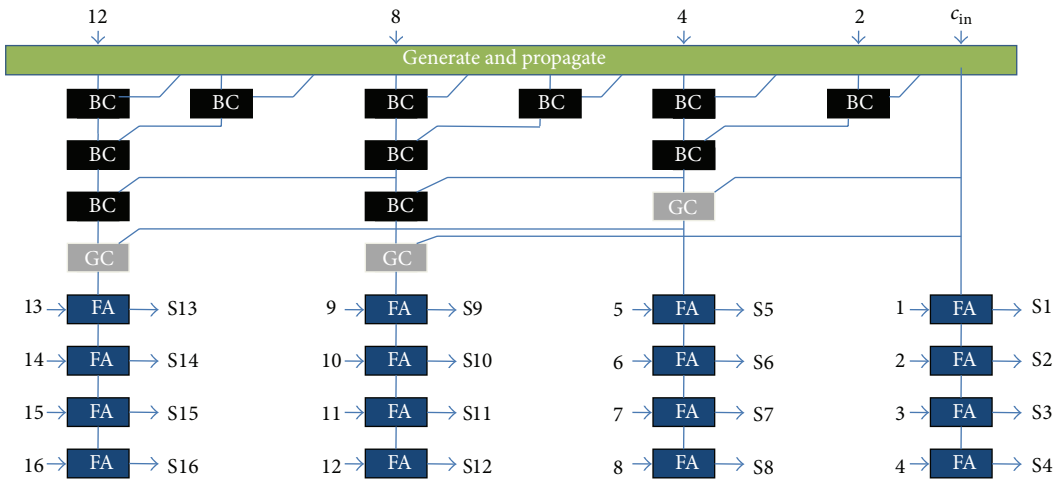

Figure 4: Illustration of a 16-bit sparse Kogge-Stone adder.

data and signal processing applications. Further details about specific FPGA architectures are documented by the various manufacturers [6, 7].

To optimize the implementation of arithmetic operations, most modern FPGAs include what is known as a fast carry chain. This improves the performance of RCAs by including a specialized connection between CLBs. This is illustrated in Figure 2 for a Xilinx FPGA. The implication is that the simple linear RCA adder will often have faster operation than an adder with a logarithmic depth, like the various tree adders discussed below.

2.2. Basic Adder Design and Performance on FPGAs. This section reviews the performance of various adder topologies implemented on FPGAs [2]. This information bears relevance for understanding the fault tolerant architectures discussed in the next section. On custom ASIC implementations, an RCA generally has $O(n)$ delay because in the worst case, the

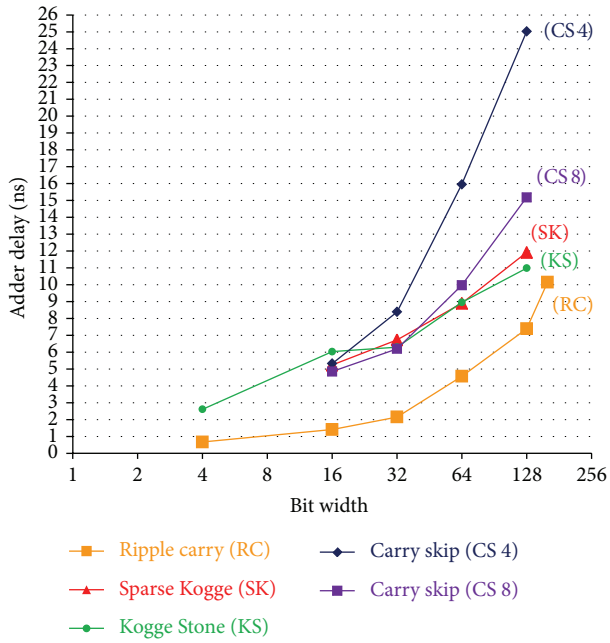

FIGURE 5: Measured results for the RCA, carry-skip adders, and Kogge-Stone adders.

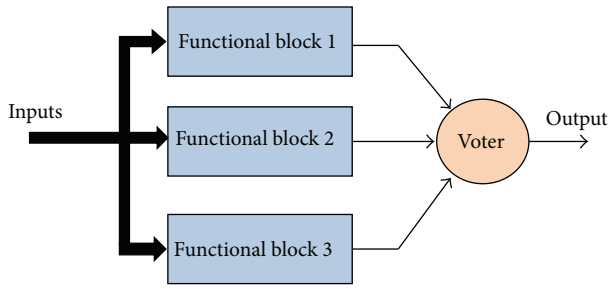

FIGURE 6: Implementation of triple mode redundancy.

carry signal must propagate through n-stages for an n-bit wide adder. A parallel-prefix adder precomputes the delay in a carry tree. The *fundamental carry operator* (*fco*), given by (1) is implemented by the logic blocks in the carry-tree [8]:

$$(g_L, p_L) \circ (g_R, p_R) = (g_L + p_L \cdot g_R, p_L \cdot p_R). \quad (1)$$

The signals g_i and p_i are the generate and propagate signals, respectively, which are generated from the input signals [9]. The associative property of the *fco* is the key to implementing a minimum logic depth carry-tree, as it allows these operators to be combined in different ways to enable the carry signals to be precomputed in parallel. For example, the carry-out signal for a four-bit adder can be implemented by the following arrangement of the *fco*, yielding a logic depth of two:

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

In general, a parallel-prefix carry tree has a logic depth $\log_2 n$ for an n-bit wide adder, and hence its delay will be on the order $O(\log n)$. The Kogge-Stone adder is one such example of a parallel-prefix adder. A further discussion on parallel-prefix adders can be found in [9, 10]. The Kogge-Stone adder is of interest for its minimal logic depth and fanout and for the spatial redundancy in the carry tree which can be exploited for fault tolerant purposes described below. The Sparse Kogge-Stone combines portions of the parallel-prefix carry-tree with a reduced set of RCAs. The two types of adders for widths of 16 bits are illustrated in Figures 3 and 4. Here we designate BC as the black cell which generates the ordered pair in (1); the gray cell (GC) generates the left signal only, following [9].

The RCA and Kogge-Stone adders along with two forms of a carry skip adder were implemented on a Xilinx Spartan 3E FPGA board and measured using a Tektronix TLA 7012 logic analyzer. The measurement methodology is described in detail in [2]. Of note is the fact that the linear RCA adder is faster than all the other adders at a bit width of 128 or less, as summarized on the graph in Figure 5. Analysis and simulation results for the Spartan 3E FPGA adders indicated for 256 bit widths that the parallel-prefix adders become faster than the linear RCAs [2].

2.3. Fault Tolerant Techniques on FPGAs. There are essentially three types of fault detection methods: (1) modular redundancy, (2) offline test methods, and (3) roving fault detection [11]. This paper will focus on the first and third methods. The second method covers test methods that occur when the FPGA is not operational, such as built-in-self-test (BIST) and I_{DDQ} test methods [12, 13].

In the first method, extra logic is added to allow concurrent error detection. The simplest example is triple mode redundancy (TMR), where a given functional block is replicated twice and a voter circuit is used to choose the majority signal, as illustrated in Figure 6. The advantage of this approach is its straightforward implementation and its relatively fast speed of detection and correction. CAD tools such as Xilinx's TMRTool exist to assist the designer in automatically creating TMR implementations on an FPGA [14]. The disadvantage of this approach is the added overhead in terms of area and power dissipation. It has been demonstrated that using TMR alone is not effective in mitigating radiation-induced single event upsets (SEUs), but utilization of the proper placement of the redundant modules can improve robustness [15]. Determining the optimal partitioning of the logic functions can also be used to improve the fault tolerance of FPGA-based systems [16]. Furthermore, combining TMR with the appropriate fault recovery system involving reconfiguration can improve robustness for multi bit upsets (MBUs) [17].

Roving fault detection involves a continuous selection of functional blocks for systematic testing that progressively scans the entire FPGA over time. A built-in-self test is performed on the selected block, which is replaced with a redundant block to allow the system to remain operational. This method has less area overhead than modular redundancy but is not quite as fast. Figure 7 illustrates the basic approach. Important work in this area includes the creation of a roving self-test area (STAR) on an FPGA, allowing the effective testing of both logic and interconnect [18, 19].

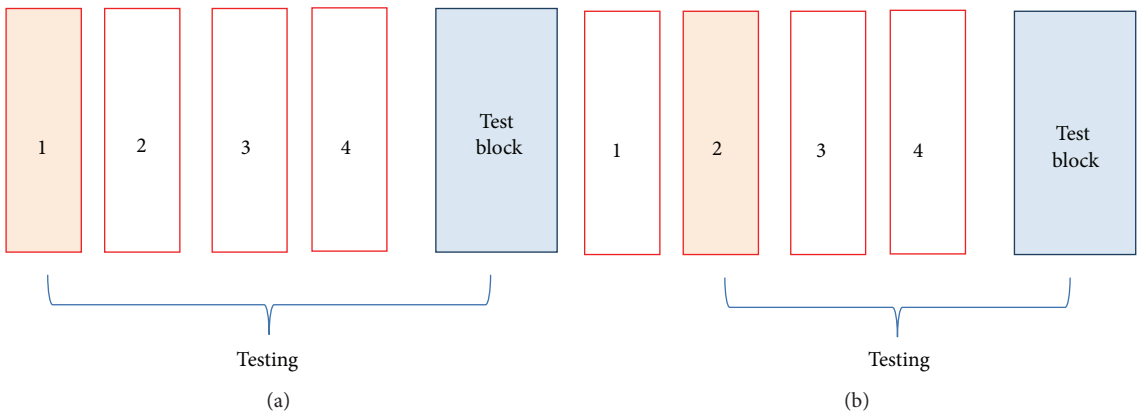

Figure 7: Illustration of the roving test method: (a) block 1 is under test, while remaining blocks remain operational, (b) block 1 returns to operation, while block 2 now undergoes testing.

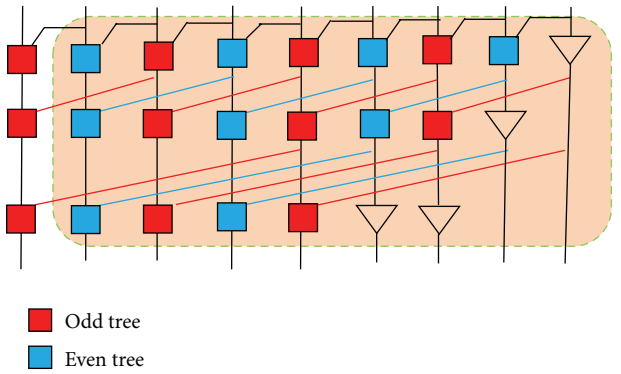

Figure 8: The carry tree of an 8-bit Kogge-Stone adder illustrating the mutually exclusive even and odd carry signals.

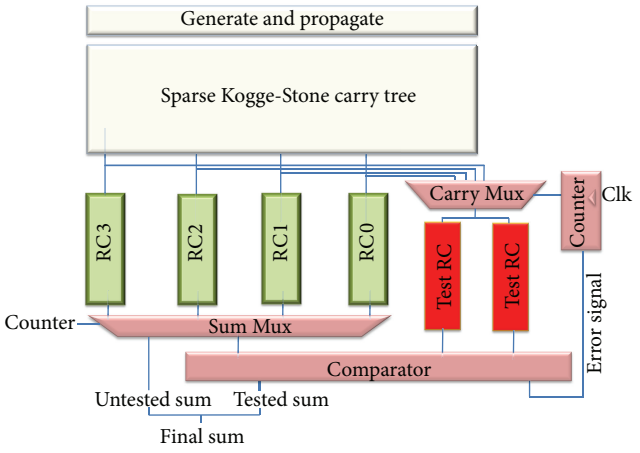

Figure 9: Block diagram of the fault tolerant sparse Kogge-Stone adder.

Fault recovery methods can also be considered to occur at three possible levels: (1) the hardware level, (2) configuration level, and (3) systems level [11]. There is some overlap between fault detection and fault recovery methods. For example, the modular redundancy approach of TMR will both detect and correct a fault. The focus of this paper will be on the hardware level for fault recovery. At this level, the FPGA remains unchanged in terms of how it is configured. This method takes advantage of the regular structure of the FPGA, allowing a faulty logic block to be replaced by a spare logic block. The most common approach at this level is to add spare columns and rows similar to what is used in large memories to improve yield. In the context of the FPGA, a spare column or row is shifted in to replace the faulty one [20].

Configuration level recovery also involves the replacement of faulty blocks with spare ones, but in this case run time reconfiguration is utilized to route in the spare blocks. There are several strategies used to determine how to reconfigure the FPGA depending on the overall goals. For example, cluster reconfiguration aims to tolerate faults which occur in clusters [21], while the goal of pebble shifting is to minimize the area and timing overhead [22]. At the system level, a spare functional module is inserted into the block. Alternatively, the faulty module is no longer used, but the modular design allows the FPGA design to continue operating, albeit at a degraded level of performance [23]. The above brief outline of fault tolerant methods for FPGAs is not meant to be exhaustive but rather to provide some context and to highlight some past approaches which are relevant for our proposed approach to fault tolerance on arithmetic logic circuits implemented on FPGAs. For comprehensive surveys on FPGA fault tolerance, see [11, 24].

3. Fault-Tolerant Adder Designs

Using modular redundancy, a fault tolerant adder on an FPGA can be efficiently constructed using three RCAs and a voter circuit. The advantage of this approach is its ease of implementation and speed. As noted in the previous section, the RCA adder is the fastest type of adder on an FPGA for bit widths at 128 or below and can be efficiently implemented due to the fast carry chain. The disadvantage of TMR is the added area and power overhead. This design will form our base implementation to compare with other designs and will be referred to as the TMR-RCA design.

A fault tolerant adder can be more compactly implemented by taking advantage of the inherent redundancy in a Kogge-Stone adder. As noted in [4], the carry tree of a Kogge-Stone adder contains mutually exclusive logic for generating

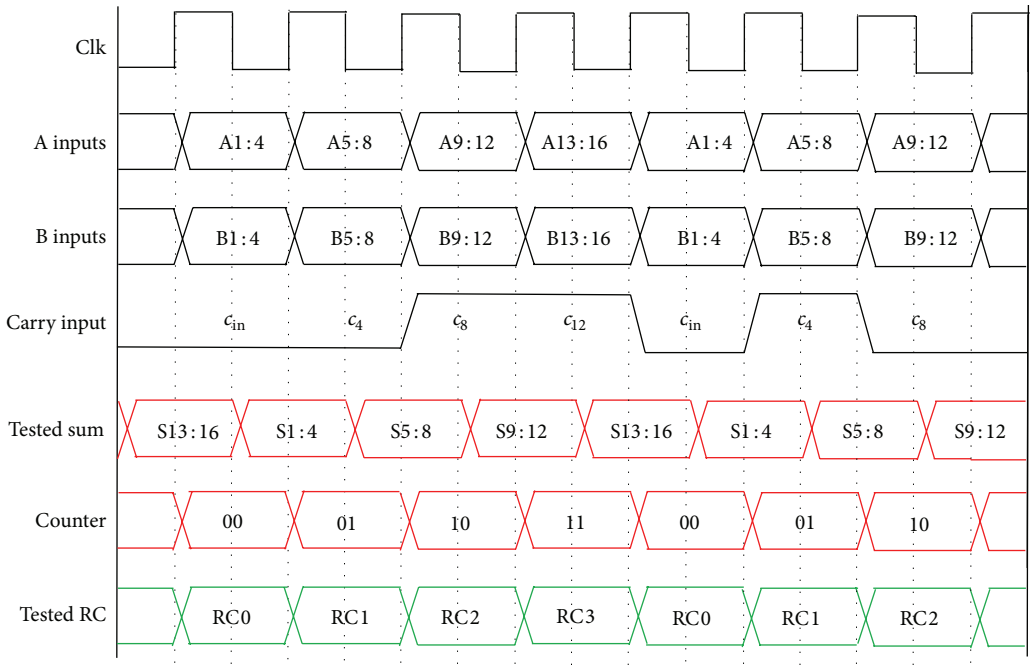

FIGURE 10: Timing diagram of the fault tolerant sparse Kogge-Stone adder.

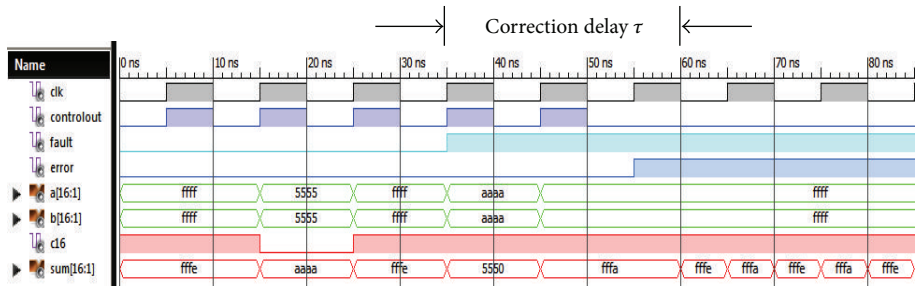

FIGURE 11: Simulation of a fault-injection into the fault tolerant sparse Kogge-Stone adder.

the even and odd carries, as illustrated in Figure 8. This spatial redundancy can be exploited to add a measure of fault tolerance to the Kogge-Stone adder as follows. If a fault occurs in one half of the tree, the other half can be utilized to compute the signals for both the even and odd carries. Essentially, there is a tradeoff in time versus area. The adder in this case will now produce a correct output once every two clock cycles instead of one. As explained in [4], with the proper scheduling and microarchitectural changes, this tradeoff can be tolerated. A drawback of this approach is that no means of fault detection was given.

This paper proposes the use of a sparse Kogge-Stone (SKS) adder to make a fault tolerant design capable of both detecting and correcting faults in the carry tree. As noted in the previous section, the SKS adder has essentially the same delay as a regular Kogge-Stone (RKS) adder when implemented on an FPGA. The carry tree is greatly simplified, although several small RCAs must be included in the design. As the RCA adder is relatively fast on an FPGA, there is no loss in speed as the simplification in the carry tree makes up for the increase in the adder's critical path. The basic idea for making the RCA section fully fault tolerant is shown in Figure 9.

The blocks labeled RC0 to RC3 are equally sized RCAs, and they can be made fault tolerant by inserting two additional RCAs, identified as TestRC in Figure 9. Both the RC0 to RC3 and TestRC blocks are identical in bit width. With the addition of some multiplexers, control circuits, and a comparator, the RCAs can be made fault tolerant in a manner similar to the TMR-RCA. During each clock cycle one of the four adders RC0 to RC3 will be selected for testing by sending its input to both of the TestRC blocks. By comparing the output of the RCA under test with the two outputs of the TestRC, a fault can both be detected and corrected. The control circuit basically consists of a counter driven by the clock. The counter drives a set of multiplexers which determine which inputs to route to the TestRC blocks. At the same time, the output of the selected RCA (one of RC0 to RC3) is connected to the voter circuit to compare its output with those of the two TestRC blocks. On the falling clock edge, the validated sum is latched and concatenated with the remaining sum bits. The timing diagram in Figure 10

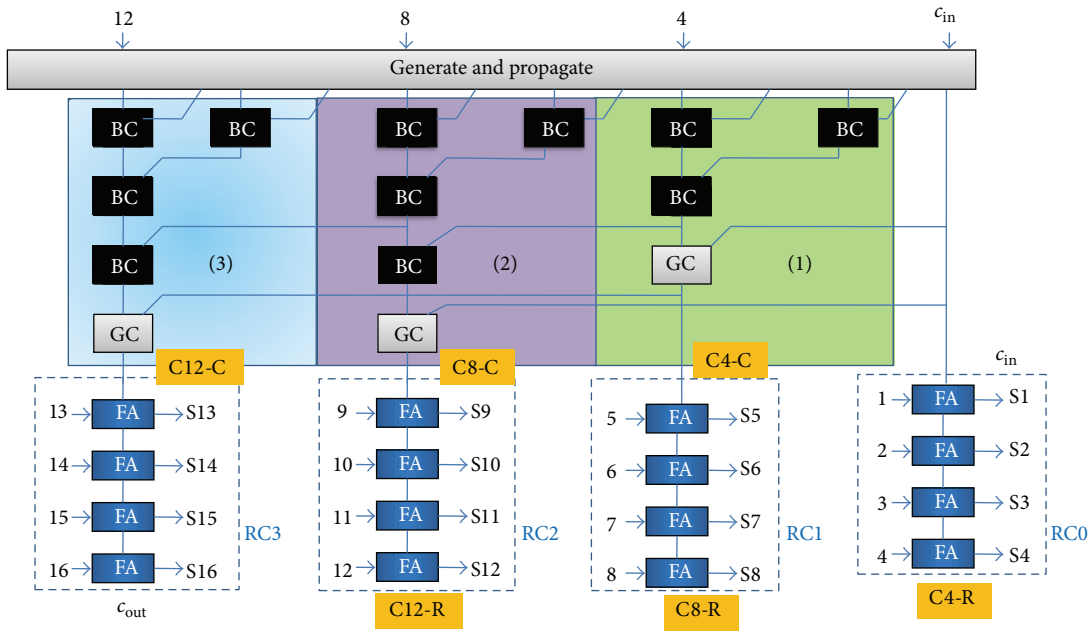

Figure 12: Fault detection scheme for the carry tree of a 16-bit sparse Kogge-Stone.

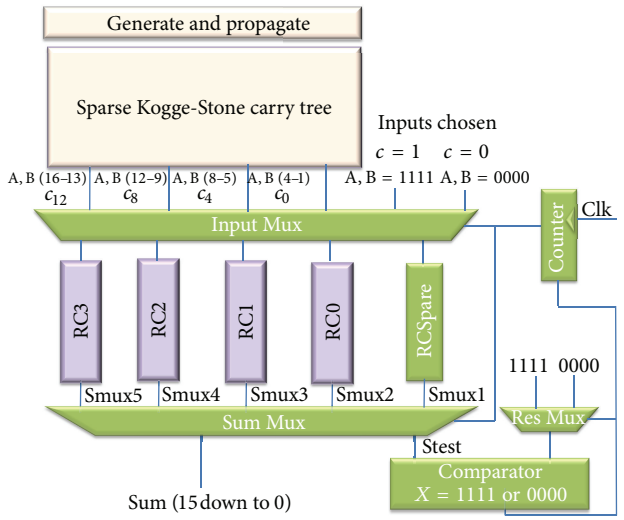

Figure 13: Implementation of graceful degradation on a sparse Kogge-Stone adder.

summarizes the operation. If an error is detected, the clock operating the bit counter is halted at the RCA where the error was detected and will continue correcting the faulty RCA until the error no longer persists.

After coding the design using VHDL, it was simulated with ISIM. Figure 11 illustrates the successful detection and correction of an error in one of the RCAs. This simulation shows a worst-case error detection and correction time delay, which occurs when the fault is introduced into an RCA immediately after it has been tested. The input signal labeled fault is used to inject a fault into one of the RCAs, and the output signal error goes high when a fault is detected. The delay τ is the time required for the error to be corrected. In the this case, three clock cycles elapse before this error is detected, which allows three incorrect results to pass through ("fffa" twice and "5550"). When the error is detected on the third clock cycle, the signal driving the counter (controlout) is held low and the correct answers are produced on the falling edge of the clock ("fffe").

Fault detection can also be implemented into the carry-tree portion of the SKS adder although a more sophisticated tradeoff using temporal redundancy is required. One approach would be to compare the carries from the carry-tree with the outputs of the RCAs, as illustrated in Figure 12. The temporal tradeoff is that the carries generated by the carry-tree (C12-C, C8-C and C4-C) will occur first as expected, followed by the carries from the RCAs (C12-R, C8-R and C4-R). By comparing the two sets of carries when they are all available, a scheme can be developed for isolating a potential fault to one of the regions labeled (1), (2), or (3) as in Figure 12. For further details on this scheme, the interested reader is referred to our discussion in [3].

Another fault tolerant scheme that can be implemented with the SKS adder is a form of graceful degradation. An RCA block, labeled RCSpare, is inserted into the design as illustrated in Figure 13. Similar to the roving method, in each clock cycle the RCSpare block is swapped with one of the RCA adders (RC0 to RC3). The swapped block then undergoes a self-test. If the swapped block is found faulty, the RCSpare block can remain permanently connected in the adder. In our particular implementation, the self-test uses one set of inputs to test the swapped RCA module, although more sophisticated tests can be applied at the cost of additional testing time. Once the RCSpare block is permanently swapped with the faulty RCA block, no further fault checking is possible and fault tolerant capability is lost.

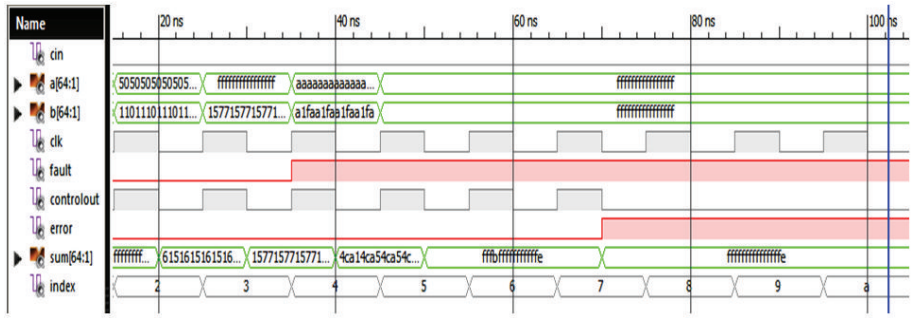

Figure 14: Simulation results for a 64-bit graceful degradation sparse Kogge-Stone adder.: (a) normal operation and (b) injection of a fault when $c_{in} = 0$, a = b = "ffffffffffffffff". The correct sum is obtained at cycle 7 ("fffffffffffffffe").

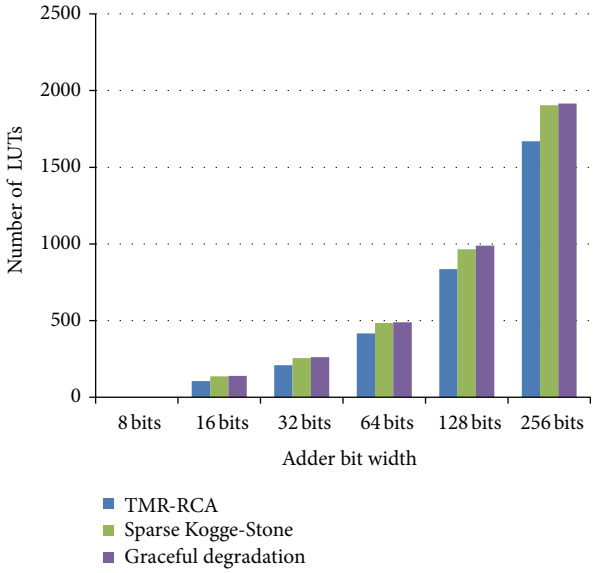

Figure 15: Amount of LUT resources used from FPGA synthesis of the adders.

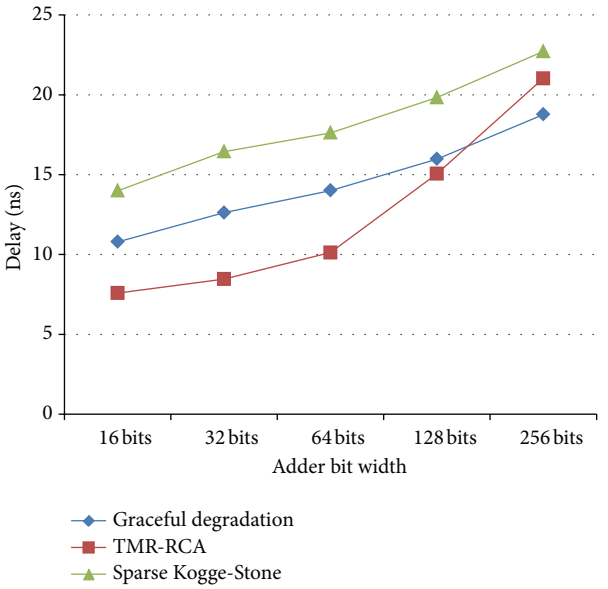

Figure 16: Delay of fault tolerant adders on a Spartan 3E FPGA from the synthesis reports.

However, this can be remedied by adding more replacement blocks. This is a tradeoff between fault tolerant capability and area.

The graceful degradation design for various bit widths was coded in VHDL and simulated using ISIM. A signal named fault is used to inject a fault into one of the RCAs, and the signal error goes high when a fault is detected. Figure 14 illustrates the simulation results when a fault is injected into one of the RCA chains for a 64-bit sparse Kogge-Stone adder using graceful degradation.

4. Experimental Results and Discussion

This section details the results from the synthesis and implementation of the three fault tolerant adder designs: the TMR-RCA, the fault tolerant sparse Kogge-Stone (SKS) adder, and the SKS adder with graceful degradation. The results are obtained for a Spartan 3E FPGA. First, a summary of the resources used on the FPGA for each adder is presented in order to give an idea of the implementation complexity. The delay characteristics for the adders of various bit widths are reported both from the synthesis reports and experimental measurements. A discussion of the results concludes this section.

Implementation statistics for the three adder were obtained by synthesizing the adders using the Xilinx *ISE* software. The number of resources in terms of look-up tables (LUTs) is reported in the graph shown in Figure 15. It can be observed that the TMR-RCA uses the lowest number of resources due to its simplicity and compact implementation made possible by the fast carry chain.

Figure 16 depicts the delay versus bit widths for the three adders as estimated from the synthesis reports from the Xilinx *ISE* tool. The TMR-RCA exhibits the best delay for bit widths of 128 and less, again due to its simple design and its ability to take advantage of the fast carry chain.

It is only at very large widths around 256 bits that the other designs become comparable in terms of delay, with the

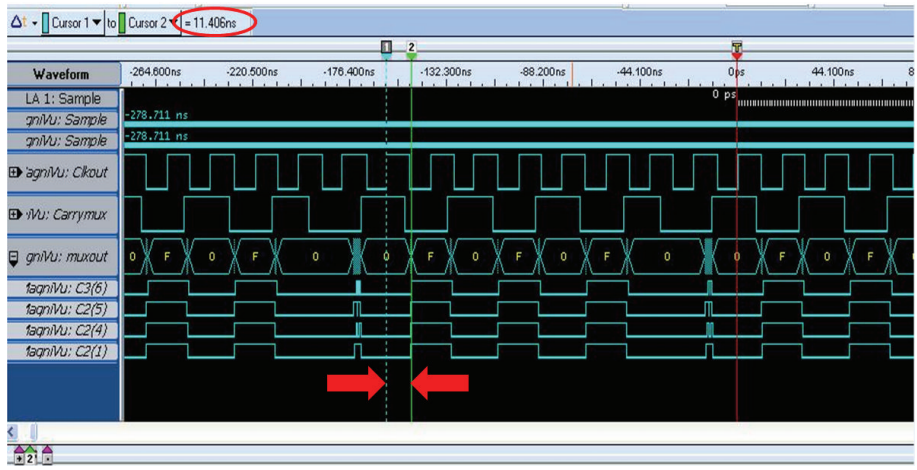

Figure 17: Screen shot of the measured delay for the 64-bit TMR-RCA using a Tektronix TLA 7012 logic analyzer. The delay obtained at a particular transition is highlighted by the red arrows.

graceful degradation approach that uses a sparse Kogge-Stone adder having the best delay at this point.

The synthesized designs were implemented on a Spartan 3E FPGA up to a width of 128 bits, and the functional verification and critical delay of the adders were determined by using a high-speed Tektronix logic analyzer (TLA 7012). The MagniVu acquisition option on our logic analyzer is able to resolve signals with a resolution down to 20 ps, allowing direct measurement of the delays. Additional test circuitry was synthesized on the FPGA to allow generation of the appropriate test signals. Our test methodology for removing delays due to cabling and the added test logic was previously reported in [2]. An example of the resulting waveforms obtained from our logic analyzer for measuring the delay is given in Figure 17.

Figure 18 summarizes the measured critical delay of the three fault tolerant adder delays as a function of adder width. While the measured delays are lower than what is predicted by the synthesis reports, the overall trend is similar: the TMR-RCA exhibits the smallest delay up to around 128 bits, followed by the graceful degradation and SKS adders.

The simple structure of the TMR-RCA and the use of the fast-carry chain give it the best delay characteristics up to very large bit widths. In addition, the entire adder chain is monitored for faults, and recovery takes place within one clock cycle. With the sparse Kogge-Stone designs, only the RCA chains are monitored for faults and several clock cycles may be needed to recover from a fault. Additional tradeoffs in terms of time and area are necessary to make the sparse carry trees also fault tolerant [3].

The results reported here indicate that the straightforward implementation of the TMR-RCA should make it the primary choice for implementing fault tolerant arithmetic logic on FPGAs unless very wide adders are being designed. However, it is worth exploring the Kogge-Stone adder further for fault tolerant implementation on ASIC designs. On a custom integrated circuit, the RCA will not have the advantage of the fast carry chain, and hence, parallel-prefix adders will have superior delay characteristics.

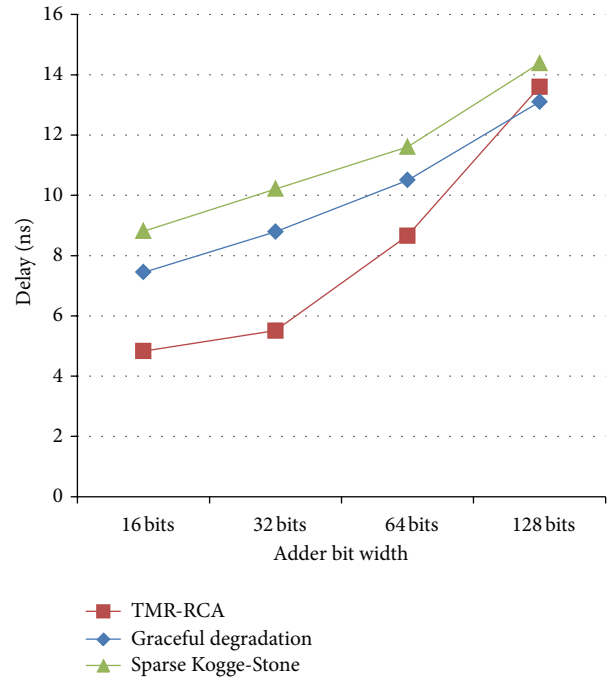

Figure 18: Measured delay of FT adders on a Spartan 3E FPGA.

5. Conclusions and Future Work

This paper has studied the implementation of fault tolerant adder designs on FPGAs using some well-known techniques, including modular redundancy, roving, and graceful degradation. Our results indicate the importance of knowing the FPGA structures and characteristics of the various adder topologies when implemented on FPGAs. For VLSI designs, parallel-prefix adders with delays on the order $O(\log n)$ have superior delay performance compared to the linear performance of the RCAs; the same does not hold for FPGA designs. The fast carry chain enables RCAs to exhibit superior delay performance on an FPGA, making the TMR-RCA the

most efficient fault tolerant design in terms of resources used and delay for FPGA implementations. But the RCA does not have this advantage on a custom IC design. Hence, it will be worthwhile to further explore the benefits of the Kogge-Stone architecture in a VLSI design. In particular, studying the tradeoffs between the length of the RCA adders and the degree of sparseness in the carry tree used in a sparse Kogge-Stone adder would be an important study for an ASIC implementation. This would allow fault detection and recovery to be accomplished with varying levels of granularity when using the roving and gradual degradation methods with the SKS adder designs. Furthermore, efficient methods for making the carry-tree fault tolerant should be studied. Our work to date indicates that a degree of fault detection and recovery can be implemented in the carry tree when additional delay and logic overhead can be tolerated. An ASIC design might have further flexibility in making this tradeoff more efficient.

References

[1] Spartan 3 Generation User Guide, http://www.xilinx.com/support/documentation/user_guides/ug331.pdf.

[2] D. H. K. Hoe, C. Martinez, and S. J. Vundavalli, "Design and characterization of parallel prefix adders using FPGAs," in *Proceedings of the 43rd IEEE Southeastern Symposium on System Theory (SSST '11)*, pp. 168–172, Auburn, Ala, USA, March 2011.

[3] C. D. Martinez, L. P. D. Bollepalli, and D. H. K. Hoe, "A fault tolerant parallel-prefix adder for VLSI and FPGA design," in *Proceedings of the 44th IEEE Southeastern Symposium on System Theory (SSST '12)*, pp. 121–125, Jacksonville, Fla, USA, March 2012.

[4] S. Ghosh, P. Ndai, and K. Roy, "A novel low overhead fault tolerant Kogge-Stone adder using adaptive clocking," in *Proceedings of the Design, Automation and Test in Europe (DATE '08)*, pp. 366–371, Munich, Germany, March 2008.

[5] P. M. Kogge and H. S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," *IEEE Transactions on Computers*, vol. 22, no. 8, pp. 786–793, 1973.

[6] Xilinx Inc, http://www.xilinx.com.

[7] Altera Corporation, http://www.altera.com/.

[8] R. P. Brent and H. T. Kung, "A regular layout for parallel adders," *IEEE Transactions on Computers*, vol. 31, no. 3, pp. 260–264, 1982.

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

[10] D. Harris, "A taxonomy of parallel prefix networks," in *Proceedings of the 37th Asilomar Conference on Signals, Systems and Computers*, pp. 2213–2217, Pacific Grove, Calif, USA, November 2003.

[11] E. Stott, P. Sedcole, and P. Y. K. Cheung, "Fault tolerant methods for reliability in FPGAs," in *Proceedings of the International Conference on Field Programmable Logic and Applications*, pp. 415–420, Heidelberg, Germany, September 2008.

[12] M. Abramovici and C. Stroud, "BIST-based detection and diagnosis of multiple faults in FPGAs," in *Proceedings of the International Test Conference*, pp. 785–794, Atlantic City, NJ, USA, October 2000.

[13] L. Zhao, D. M. H. Walker, and F. Lombardi, "I_{DDQ} testing of bridging faults in logic resources of reconfigurable field programmable gate arrays," *IEEE Transactions on Computers*, vol. 47, no. 10, pp. 1136–1152, 1998.

[14] Xilinx TMR Tool, http://www.xilinx.com/ise/optional_prod/tmrtool.htm.

[15] L. Sterpone and M. Violante, "Analysis of the robustness of the TMR architecture in SRAM-based FPGAs," *IEEE Transactions on Nuclear Science*, vol. 52, no. 5, pp. 1545–1549, 2005.

[16] F. L. Kastensmidt, L. Sterpone, L. Carro, and M. S. Reorda, "On the optimal design of triple modular redundancy logic for SRAM-based FPGAs," in *Proceedings of the Design, Automation and Test in Europe (DATE '05)*, pp. 1290–1295, Munich, Germany, March 2005.

[17] M. G. Gericota, L. F. Lemos, G. R. Alves, and J. M. Ferreira, "On-line self-healing of circuits implemented on reconfigurable FPGAs," in *Proceedings of the 13th IEEE International On-Line Testing Symposium (IOLTS '07)*, pp. 217–222, Crete, Greece, July 2007.

[18] M. Abramovici, J. M. Emmert, and C. E. Stroud, "Roving STARs: an integrated approach to on-line testing, diagnosis, and fault tolerance for FPGAs in adaptive computing systems," in *Proceedings of the 3rd NASA/DoD Workshop on Evolvable Hardware*, pp. 73–92, Long Beach, Calif, USA, July 2001.

[19] J. M. Emmert, C. E. Stroud, and M. Abramovici, "Online fault tolerance for FPGA logic blocks," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 15, no. 2, pp. 216–226, 2007.

[20] F. Hatori, T. Sakurai, K. Nogami et al., "Introducing redundancy in field programmable gate arrays," in *Proceedings of the IEEE Custom Integrated Circuits Conference (CICC '93)*, pp. 7.1.1–7.1.4, San Diego, Calif, USA, May 1993.

[21] V. Lakamraju and R. Tessier, "Tolerating operational faults in cluster-based FPGAs," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '00)*, pp. 187–194, Monterey, Calif, USA, February 2000.

[22] J. M. Emmert and D. Bhatia, "Partial reconfiguration of FPGA mapped designs with applications for fault tolerance and yield enhancement," in *Proceedings of the 7th International Workshop on Field Programmable Logic and Applications*, vol. 1304, pp. 141–150, Lecture Notes in Computer Science, London, UK, September 1997.

[23] Y. Nakamura and K. Hiraki, "Highly fault-tolerant FPGA processor by degrading strategy," in *Proceedings of the 2002 Pacific Rim International Symposium on Dependable Computing*, pp. 75–78, Tsukuba, Japan, December 2002.

[24] J. A. Cheatham, J. M. Emmert, and S. Baumgart, "A survey of fault tolerant methodologies for FPGAs," *ACM Transactions on Design Automation of Electronic Systems*, vol. 11, no. 2, pp. 501–533, 2006.