Voter Insertion Techniques for Fault Tolerant FPGA Design

Jonathan Johnson
Michael Wirthlin
NSF Center for High Performance
Reconfigurable Computing (CHREC)
Dept. of Elec. & Comp. Engineering
Brigham Young University
Provo, UT 84604, USA
jonjohn@byu.net, wirthlin@ee.byu.edu

Abstract—Triple Modular Redundancy (TMR) is a common reliability technique for FPGA designs used in radiation environments. TMR consists of triplicating a design and inserting voters to mask errors using redundancy. This paper will investigate the automatic placement of voters in TMR designs. In particular, it will introduce three algorithms for determining where to insert synchronization voters and compare the area and timing impact of these algorithms on FPGA designs. It will be shown that the placement of synchronization voters in a triplicated design can have an important impact on the area and timing characteristics of the resulting design. The algorithms presented in this paper give results that increase the critical path length of a design when adding TMR voters by as little as 3% to as much as 50%.

TABLE OF CONTENTS

1 INTRODUCTION 1
2 FPGA SINGLE EVENT EFFECTS 1
3 MITIGATION APPROACHES 2
4 TMR VOTER INSERTION 3
5 SYNCHRONIZATION VOTER INSERTION ALGORITHMS 5
6 EXPERIMENTAL COMPARISON OF ALGORITHMS 6
7 CONCLUSIONS 7
REFERENCES 8

1. INTRODUCTION

SRAM-based FPGAs are an attractive alternative to ASICs in space-based computing missions for several reasons. Their reconfigurability allows them to be used to perform various tasks at different times during a mission. FPGAs are often used to implement custom designs that attain application specific performance that would not be possible with software reconfigurable only alternatives. In addition, the use of FPGAs can reduce the overall non-recurring engineering (NRE) costs involved in developing a space-based application [1], [2], [3], [4].

This work was supported by the Rocky Mountain Space Grant Consortium. This work was also supported by the I/UCRC Program of the National Science Foundation under Grant No. 0801876.

SRAM-based FPGAs are, however, susceptible to radiation effects in space environments. The functionality of an FPGA is dependent on the integrity of its configuration memory. FPGA configuration memories are very large and are susceptible to errors caused by single event upsets (SEUs).

Triple Modular Redundancy (TMR) is the most commonly used mitigation technique against SEUs for FPGA designs used in radiation environments. The basic concept of TMR is to triplicate a circuit design so that the resulting design consists of three redundant copies of the original. Majority voters are used to mask errors in any single copy of the circuit.

The insertion of voters is an important aspect of applying TMR to a design. One of the more challenging design problems is deciding where to insert synchronization voters. Synchronization voters are used to keep the sequential logic state of the three redundant copies of a circuit (domains) synchronized when there are SEUs that affect design feedback. In order to keep a design synchronized properly, synchronization voters must be inserted in enough locations to intersect all of the feedback in a design.

TMR is often implemented by hand, and the process of properly inserting synchronization voters manually can be tedious and error-prone. This paper will introduce three different algorithms for automatically determining suitable synchronization voter insertion locations. These algorithms have been implemented in an automated TMR tool developed at Brigham Young University in collaboration with Los Alamos National Laboratory [5]. While all three algorithms properly intersect all design feedback with synchronization voters, they do so in different ways. The choice of where to insert synchronization voters affects how many voters are needed as well as the critical path length of a design. This paper will compare the three synchronization voter insertion algorithms and show that the choice of voter insertion locations has a significant impact on both circuit area and timing performance.

2. FPGA SINGLE EVENT EFFECTS

The voter insertion algorithms presented in this paper are motivated by FPGA implementations of the TMR technique.

This section will summarize single event effect (SEE) issues in FPGAs that make techniques such as TMR necessary in FPGA designs for space-based missions. While FPGA manufacturers generally guarantee the total ionizing dose (TID) life and single event latchup (SEL) immunity of their radiation hardened devices [6], the devices are susceptible to single event upsets (SEUs). In an SRAM-based FPGA, an SEU occurs when a charged particle strikes an SRAM cell, causing the state of the memory cell to change. SEUs are problematic for FPGAs because their configuration memories contain millions of memory cells which makes them a large target for SEUs.

The functionality of an FPGA is dependent on the contents of its configuration memory. FPGAs are typically made up of highly configurable logic blocks containing lookup tables (LUTs) that define logic functions and registers used for sequential logic. A reconfigurable routing network connects the logic blocks in an FPGA in order to implement complex designs. The contents of LUTs, the functionality of registers, and the routing network connections are all stored in an FPGA's configuration memory. The functionality of an FPGA changes when the contents of its configuration memory change.

An SEU in an FPGA's configuration memory affects only a single bit of memory. However, a single bit flip can have significant consequences on FPGA functionality. For example, a single bit flip in a LUT can change a boolean AND function to a boolean function that always outputs a logical 0. A single bit flip can also change the connections in the FPGA's routing network. The results of an SEU in an FPGA's configuration memory can be unpredictable.

FPGAs are also sensitive to SEUs in Block RAMs (BRAMs) and user flip-flops. BRAMs are often used as memories or FIFOs in FPGA designs. User flip-flops are the registers in the FPGA that are instantiated in a design for use in state machines, counters, and other sequential logic structures. BRAM and user flip-flop upsets can cause a design to enter invalid states. Although these kinds of upsets are important, the configuration memory has a much larger cross section and is more likely to receive SEUs.

3. MITIGATION APPROACHES

The most common mitigation approach for FPGAs used in radiation environments is a combination of bitstream scrubbing and TMR. Bitstream scrubbing corrects errors in an FPGA's configuration memory after they occur and TMR masks circuit functionality errors as they occur. When used together, the two techniques improve reliability significantly.

Bitstream Scrubbing

SEUs in the configuration memory of an FPGA can be corrected by bitstream scrubbing [7], [8]. In bitstream scrubbing, the FPGA's configuration control logic is used to periodically read the configuration memory and check for errors using a pre-computed CRC code. Upon detection of an SEU-induced error, partial reconfiguration is used to repair the affected contents of the configuration memory. In this manner, bitstream scrubbing corrects errors in the configuration memory soon after they occur.

Although errors are corrected quickly, there is a finite amount of time between the occurrence of an upset and the partial reconfiguration that corrects it. During this time, circuit functionality modified by the upset can cause errors in computation. Such errors can propagate to circuit outputs or feed back into sequential logic state, causing incorrect circuit operation to persist even after the effect of the SEU on the configuration memory is corrected. In addition, bitstream scrubbing cannot address errors caused by SEUs in BRAMs and user flip-flops because the correct value of the corresponding memory cells cannot be known unless redundancy is employed. Bitstream scrubbing becomes a more effective mitigation approach when used in conjunction with an error masking technique such as TMR.

Triple Modular Redundancy

TMR is a well known technique for improving the reliability of integrated circuits. Three redundant copies of a circuit are created and majority voters are used to mask errors that occur in any of the three copies (see Figure 1).

Although TMR is often applied to designs manually, the process is straightforward enough to be implemented by an automated CAD tool. Existing tools for applying TMR to FPGA designs include the Xilinx XTMR tool [9] and the BLTmr tool developed at Brigham Young University in collaboration with Los Alamos National Laboratory [5]. Using an automated tool can provide several advantages over implementing TMR by hand. For example, inserting voters in the proper places manually can be a tedious and error prone process. Another pitfall when attempting to implement TMR manually is that many synthesis tools remove redundant logic. This issue is avoided by automated CAD tools that operate on a post-synthesis circuit representation (i.e. EDIF netlist). One further advantage of the BLTmr tool in particular is that it allows for prioritized partial triplication of designs based on target device size [10].

In a TMR system, majority voters are used to mask errors that occur in any single copy of the circuit. In FPGAs, voters are most commonly implemented using look-up tables. A three input look-up table (LUT3) is sufficient to implement a single majority voter. In general, triplicated voters are used to avoid single points of failure. As shown in Figure 1, each of the three voters takes an input from each domain and passes outputs along to its respective domain.

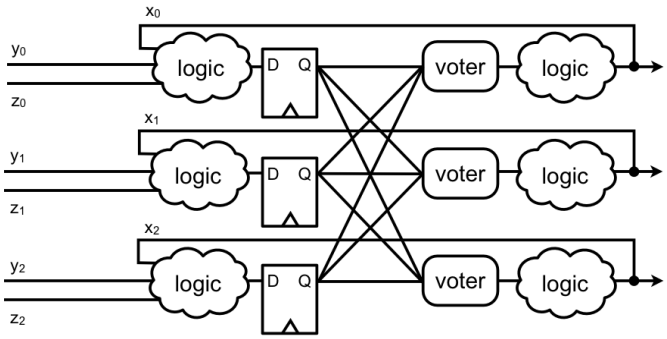

Figure 1. Triplicated Voters

Reliability Modeling

Reliability modeling is important for determining the benefits of using various reliability techniques. FPGA reliability can be modeled using several methods, including combinatorial modeling [11] and markov modeling [12]. In general, a non-redundant system is less reliable than a TMR system without repair (i.e. without bitstream scrubbing) for short mission times. Overall, a TMR system with bitstream scrubbing is much more reliable than both a non-redundant system and a TMR system without repair. Figure 2 compares the reliability of three example systems with these configurations using combinatorial and markov modeling techniques.

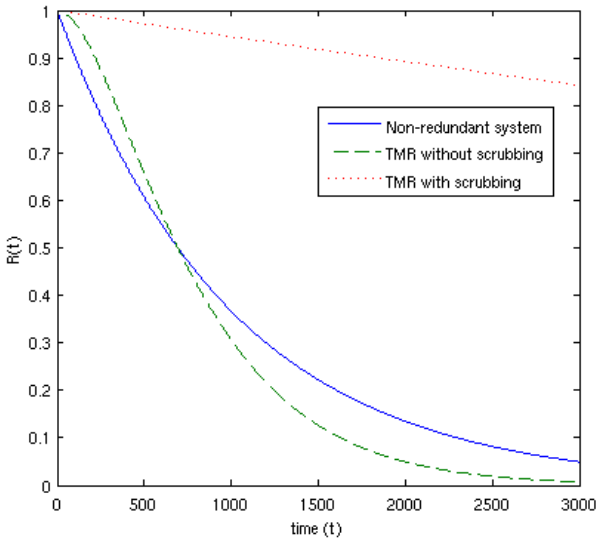

Figure 2. Reliability Comparison for TMR

4. TMR Voter Insertion

While the basic concept of TMR is straightforward, determining where to insert voters can be somewhat difficult. In general, the reasons for inserting voters suggest where they should be placed. For example, reducing voters are used to reduce a signal from three domains to a single domain (generally at circuit outputs) and clock domain crossing voters mitigate TMR vulnerabilities created by clock domain crossing synchronizers. Optimal locations for other types of voters can be more difficult to determine. Partitioning voters subdivide a circuit into TMR partitions for higher reliability. Determining the optimal number and locations of partitions is difficult. Synchronization voters keep the sequential logic state of TMR domains synchronized when upsets in feedback sections of a design occur. Determining optimal locations for synchronization voters is also difficult.

Reducing Voters

Reducing voters take outputs from three separate TMR domains as input and produce a single output. The most common use of reducing voters is at circuit outputs. Sometimes it is desirable to have a single set of circuit outputs rather than output all three TMR domains for external voting. This can be necessary, for example, when the target FPGA has insufficient I/O resources to allow full triplication of the circuit outputs. In such a situation, reducing voters are used to reduce three TMR domains to a single output as shown in Figure 3.

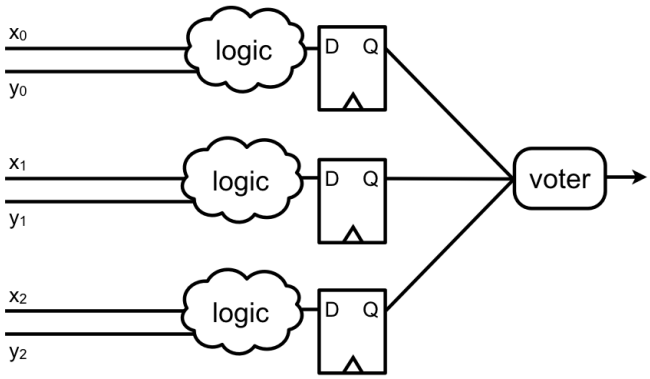

Figure 3. Reducing Voter

Reducing voters are also useful in partial TMR configurations. When partial TMR is used, there are circuit locations where data must flow from a triplicated partition to a non-triplicated partition. Reducing voters are used at these locations to provide a single input to the non-triplicated partition.

TMR can also be mixed with duplication with compare (DWC), an error detection technique which uses duplication instead of triplication. In such a configuration, there are circuit locations where data must flow from a triplicated circuit partition to a duplicated partition. At such locations, two reducing voters are used in parallel to reduce the three TMR domains to two inputs for the duplicated partition.

Voter Partitioning

In a typical TMR system, errors that occur in the configuration memory are discovered and corrected by scrubbing. In a circuit that has voters only at the outputs, errors are masked

as long as they occur in only one of the three TMR domains at a time. If multiple errors occur fast enough such that they accumulate in more than one domain before being corrected by scrubbing, the redundancy is overcome and errors can reach circuit outputs. This vulnerability can be mitigated by subdividing the circuit into multiple partitions and applying TMR to each partition separately. The partitions are separated with triplicated voters.

In a TMR system with multiple partitions, each partition can tolerate errors in a single domain. That is, the system can tolerate concurrent non-overlapping failures (failures in separate partitions and possibly separate domains). The reliability of the circuit can be improved by subdividing it into smaller and smaller partitions up to the point where the reliability gains from partitioning are overridden by the unreliability of the voters being added in between the partitions.

Clock Domain Crossing Voters

Special consideration is required when applying TMR to circuits with multiple clock domains. Clock domain crossing synchronizers are a TMR domain synchronization hazard. A typical clock domain crossing synchronizer consists of a number of consecutive flip-flops to reduce the probability of a metastable value propagating through the entire synchronizer. Because of the uncertainty associated with metastability, three synchronizers whose inputs transition at exactly the same time do not necessarily propagate the correct output at the same time. This means that TMR domains have the potential to be unsynchronized after clock domain crossing synchronizer outputs even without radiation effects. The problem is compounded by the fact that the inputs to the clock domain crossing synchronizers are three separate nets with three separate timing paths. Even the synchronizer inputs may not transition at exactly the same time.

The problem that is created by TMR domains being possibly unsynchronized after clock domain crossing synchronizers is that it leaves the circuit vulnerable to SEUs. When only two of the domains are synchronized, a single error in either of them can cause an incorrect output that is not masked by TMR. This is because a single TMR partition can tolerate errors in only a single domain at a time.

Several strategies for mitigating TMR circuits with clock domain crossings are being investigated. These strategies involve strategically placing voters in order resynchronize the TMR domains after the clock domain crossing syncrhonizers.

Synchronization Voters

Synchronization voters are used to keep the state of the three TMR domains in a triplicated design synchronized in the face of SEUs. Consider a simple triplicated counter with voters inserted only at design outputs (see Figure 4). If an error were to occur in one of the domains, it would have the potential to cause incorrect values to feed back into the state of the counter in the domain in error. This domain would then continue to produce incorrect results even if the original error in the circuit functionality were corrected via bitstream scrubbing. Such an error is called a persistent error [13] because the state of the affected domain remains unsynchronized even after the original error in the configuration memory is corrected. One way to recover from a persistent error is to reset the design after the configuration memory has been corrected. Resetting the design frequently is costly in terms of system availability.

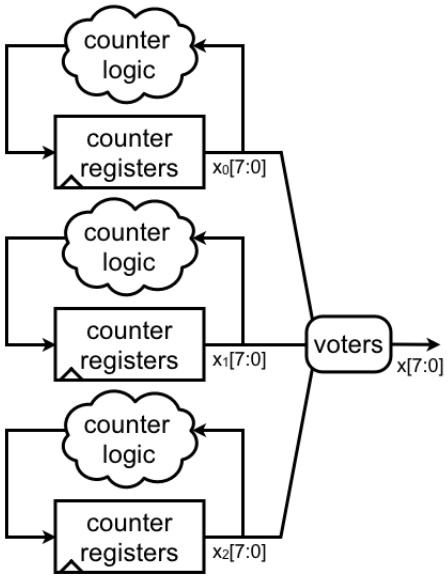

Figure 4. Simple Counter

Synchronization voters provide a better solution than frequently resetting a design. Consider the same triplicated counter as before but this time with triplicated voters inserted in the feedback path (see Figure 5). The added voters mask errors that would normally reach and remain in the state of the affected domain. Errors that occur after the voters (i.e. in the counter logic) can still cause incorrect values to reach the counter registers, but they will be flushed out after the configuration memory error is corrected via bitstream scrubbing, and the voters keep the errors from ever reaching the rest of the design. Triplicated voters placed in design feedback paths in this manner are called synchronization voters because they keep the state of the three domains from becoming permanently unsynchronized by errors that would otherwise affect design state in a persistent manner.

The placement of synchronization voters is the most difficult voter placement issue to resolve automatically. Synchronization voters should be placed at locations that cut all design feedback, but there are many ways to cut all of the feedback in a design. In addition, synchronization voters can slow down a design's critical timing path. Algorithms for determining appropriate places to insert syncrhonization voters are discussed in the next section.

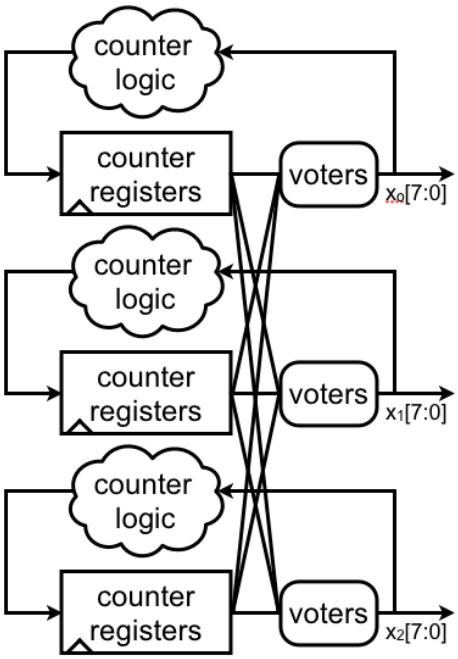

Figure 5. Synchronized Simple Counter

5. Synchronization Voter Insertion Algorithms

This section will present three different algorithms for determining where to insert synchronization voters. The insertion of synchronization voters presents a particular challenge, especially for FPGA designs. In order to effectively mitigate against persistent errors, all of the feedback paths in a design must be cut by triplicated voters. In a general sense, this problem can be considered an instance of the feedback edge set (FES) problem in which one is given a directed graph and asked to find a minimum subset of edges that intersects every cycle in the graph. The unweighted version of the FES problem is NP-hard [14].

It is difficult to determine the best locations to insert synchronization voters because the choice has a significant impact on the resulting circuit's timing performance and area and because there are certain locations where voters cannot be inserted due to FPGA architectural constraints. When multiple voters are placed in a single timing path, the performance of the circuit is adversely affected. Voter insertion locations also determine the total number of voters needed to cut all feedback and hence affect circuit area. The three algorithms presented in this section were implemented in the BLTmr tool at Brigham Young University. They are the basic SCC decomposition algorithm, the highest fanout SCC decomposition algorithm, and the highest flip-flop fanout SCC decomposition algorithm. Their impact on area and timing will be discussed in section 6.

Basic SCC Decomposition

The first algorithm considered is a simple decomposition of strongly connected components (SCCs). The circuit netlist is first converted to graph form and analyzed to find SCCs. Then, the SCCs are decomposed into successively smaller SCCs by removing feedback edges until there is no feedback left in the circuit graph. Special consideration is made for circuit locations where voters cannot be inserted due to architectural constraints. These locations are referred to as bad cut edges. Pseudocode for the algorithm is given in Algorithm 1.

Algorithm 1 Basic SCC Decomposition Algorithm

```
Initialize list L
Initialize stack S
Perform SCC analysis
Push SCCs onto S
while S not empty do
  current_scc = S.pop()
  Find back edges in current_scc
  if back edges are all legal cuts then
    Remove all back edges from SCC
    Save removed back edges to L
  else if back edges are both bad and legal cuts then
    Remove only legal cut back edges
    Save removed back edges to L
    Clear S
    Recompute SCC analysis
    Push SCCs onto S
  else
    Clear S
    Perform SCC analysis with new vertex visit order
    Push SCCs onto S
  end if
end while
Insert voters on edges in L
```

The basic SCC decomposition algorithm correctly mitigates against persistent errors by inserting voters that cut all design feedback, but no attempt is made to optimize the voter locations to minimize the impact of voters on circuit area or timing.

Highest Fanout SCC Decomposition

The highest fanout SCC decomposition algorithm is an attempt to reduce the number of voters used to cut feedback. It is based on the fact that many circuit designs have some high fanout nets. When a voter is inserted on a net with high fanout, a significant amount of feedback can be cut with only a single voter. The highest fanout SCC decomposition algorithm takes advantage of this by decomposing SCCs in a manner that prioritizes the removal of the edges that can cut the most feedback with the fewest voters. That is, each SCC is analyzed to find the node with the highest legal cut fanout and its legal cut output edges are cut first. Pseudocode for this algorithm is given in Algorithm 2.

Algorithm 2 Highest Fanout SCC Decomposition Algorithm

```
Initialize list L
Initialize stack S
Perform SCC analysis
Push SCCs onto S
while S not empty do
  current_scc = S.pop()
  Node n = Find node with highest legal cut fanout
  Remove from graph the legal cut edges coming from n
  Recompute SCC analysis of current_scc subgraph
  Push SCCs onto S
  Save removed edges to L
end while
Insert voters on edges in L
```

It will be shown in section 6 that the highest fanout SCC decomposition algorithm is indeed effective at reducing the number of voters used but that it does not always provide better timing results than the basic SCC decomposition algorithm.

Highest Flip-Flop Fanout SCC Decomposition

The highest flip-flop fanout SCC decomposition algorithm is designed to both reduce the number of voters used to cut feedback over the basic SCC decomposition algorithm and minimize the impact of the voters on circuit timing. Voters can negatively affect timing more than is necessary when more than one set of voters is placed in a single path from one register to the next. This algorithm prevents this from happening by decomposing SCCs as before but by prioritizing the removal of edges coming from flip-flop nodes with high legal cut fanouts. Since a timing path consists of the logic from one flip-flop to the next, inserting voters directly after flip-flops ensures that only one voter will be inserted per timing path. The pseudocode for this algorithm is the same as that of the previous algorithm except that for each SCC the algorithm finds the *flip-flop* node with the highest legal cut fanout instead of the node with the highest legal cut fanout overall.

6. Experimental Comparison of Algorithms

This section will present the results of applying TMR to a suite of test designs using each of the three synchronization voter insertion algorithms described in the previous section. Both area and timing impact will be considered.

Designs

A suite of test designs including both real world and synthetic designs was selected to test the effectiveness of the three synchronization voter insertion algorithms. Only designs that include some amount of feedback were selected since synchronization voters are unnecessary in feed forward only designs.

The *MACFIR* design implements a multiply accumulate (MAC) unit using a feedback loop. A MAC unit performs a sum-of-products operation that is useful for computing a convolution sum. Such a design can be used to implement a FIR (finite impulse response) filter for signal processing applications.

The *DES3* design implements a triple DES encrypter. Triple DES is a block cipher used in cryptography applications. It uses three keys and works by first encrypting data using the first key, decrypting the data with the second key, and finally encrypting the data with the third key. This design was chosen because it is a computationally intensive real world application.

The *QPSK* design is a quadrature phase-shift keying (QPSK) demodulator. QPSK is a digital modulation scheme used in communications applications in which data is encoded using the phase of the carrier signal. This design contains a fair amount of feedback and is another computationally intensive real world application.

The *Synthetic* design is a design that was crafted to contain both feedback and feed forward logic. It consists of a linear feedback shift register (LFSR) whose output is combined with an input signal using a multiplier and an adder tree. While it is not necessarily a real world application, it is useful because it contains feedback (making synchronization voters necessary) and uses a large portion of the resources available on the target FPGA device. This is interesting because it results in routing congestion which makes it more difficult for the place and route software to find a routing that meets timing constraints.

The *LFSRs* design is another synthetic design that consists of a large LFSR replicated ten times. It is interesting because it contains a large amount of feedback, and the feedback inherent in an LFSR is of a fairly complex nature, meaning that there are many possible synchronization voter configurations for cutting the feedback.

Test Procedure

Each of the test designs was triplicated using the automated TMR tool developed at Brigham Young University. Each of the three synchronization voter insertion algorithms was used on each design. In all of the designs except the *DES3* design, full triplication of all circuit elements and I/Os except the clock was performed. There is a large amount of I/O in the *DES3* design, and there were insufficient I/O resources on the target FPGA device to facilitate full triplication of the inputs and outputs so they were left untriplicated. In each iteration of each design, the total number of voters inserted into the design was recorded.

The target FPGA device for these experiments is the Xilinx Virtex 1000 (XCV1000-5-bg560). Using a script to control the Xilinx tool flow, a place and route was performed on each iteration of each design using successively tighter timing con-

		MACFIR	QPSK	DES3	Synthetic	LFSRs
Original Design	Critical Path	14.7 ns	79.8 ns	10.9 ns	9.8 ns	11.9 ns
Basic SCC Decomposition	Critical Path	18.6 ns	120 ns	16.2 ns	10.5 ns	13.6 ns
	Total Voters	219	1188	543	720	570
Highest Fanout Decomposition	Critical Path	18.5 ns	90.2 ns	14.9 ns	11.2 ns	13.7 ns
	Total Voters	219	165	434	288	360
FF Fanout Decomposition	Critical Path	18.3 ns	84.3 ns	13.6 ns	10.9 ns	12.3 ns
	Total Voters	219	96	352	324	450

Table 1. Synchronization Voter Insertion Algorithm Comparison Results

straints until the place and route run failed to find a routing capable of meeting the timing constraint. In this manner, the best possible critical path length for each iteration of each design was determined. Timing constraints were adjusted in 0.1 ns increments.

Results

The results of the synchronization voter insertion algorithm comparison experiments are given in Table 1. An analysis of the results provides some useful insights about synchronization voter insertion.

First, it is interesting to compare the number of voters produced by each algorithm. The basic SCC decomposition algorithm consistently gives the highest number of voters. The lowest number of voters is given by the highest flip-flop fanout SCC decomposition algorithm in the *QPSK* and *DES3* designs (two of the real world designs) while the highest fanout SCC decomposition algorithm gives the lowest number of voters in both of the synthetic designs (*Synthetic* and *LFSRs*). This could be due to the fact that the *Synthetic* and *LFSRs* designs both have similar feedback patterns (due to LFSRs) that are distinct from the feedback patterns in the real world designs. This suggests that an adaptive algorithm based on analyzing the type of feedback in a design could be used for more effectively minimizing the number of voters in a triplicated design when that is the primary focus.

Given the relatively low impact of voters on circuit area, it is generally more likely that timing is the primary concern when inserting synchronization voters. In all designs except *Synthetic*, the best timing results are obtained by using the highest flip-flop fanout SCC decomposition algorithm. This is an expected result based on the fact that the highest flip-flop fanout SCC decomposition algorithm is designed to minimize the number of voters placed in a single timing path. Unexpectedly, the basic SCC decomposition algorithm produces the best timing results for the *Synthetic* design. This could be an anomaly due to the fact that the *Synthetic* design uses a large portion of the target FPGAs resources, creating routing congestion. Such congestion makes it much more difficult for the place and route software to produce a routing that meets timing constraints. In such conditions, place and route results are often more arbitrary.

It is interesting to note that it is not always the case that the best timing results are obtained by the algorithm that produces the lowest number of voters. The results for the two synthetic designs illustrate this. This could be due to the nature of the feedback in these designs. Since the feedback in both is due to LFSRs, which have a complex feedback pattern, there are many possible ways to cut all of the design feedback. In a design where feedback is due to a simple counter or state machine structure, cutting all of the feedback is a simpler problem because the feedback loops are simpler.

Finally, the case of the *MACFIR* design is particularly interesting because all three algorithms produced the same number of voters, but the placement of the voters gave different timing results for each algorithm. The highest flip-flop fanout SCC decomposition algorithm gave the best timing results, suggesting that the heuristic of inserting voters directly after flip-flops employed by this algorithm is a good way to determine voter locations.

7. CONCLUSIONS

Because of its impact on area and timing performance, voter insertion is an important issue when using TMR in FPGA designs. Three algorithms for inserting synchronization voters have been described and compared based on experimental results. The highest flip-flop fanout SCC decomposition algorithm provides the best results overall in terms of both area and timing impact on the designs considered in this paper. On average, the increase in critical path length due to inserted voters with this algorithm was 23% better than the increase given by the best of the other two algorithms; the average number of voters inserted by the algorithm was 5% less than the best of the other two algorithms.

The problem of deciding where to insert voters to cut all design feedback is an instance of the feedback edge set problem (FES), which is NP-hard. Algorithms exist for approximating the weighted version of this problem [15]. One possible direction for future work could be to cast the problem of where to insert synchronization voters as an instance of the weighted FES problem using appropriate edge weights to try to minimize circuit area and timing impact. Existing approximation algorithms could be used to provide solutions for such a problem.

TMR with bitstream scrubbing is an effective reliability technique for FPGA designs used in space-based missions. Inserting voters to ensure reliability can be a hard problem when implementing TMR manually, but the highest flip-flop fanout SCC decomposition algorithm presented in this paper provides an effective way of inserting voters automatically with a low impact on circuit area and timing performance.

REFERENCES

[1] D. Ratter, "FPGAs on Mars," Xilinx, Tech. Rep., August 2004, xCell Journal #50.

[2] M. Caffrey, "A space-based reconfigurable radio," in *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, T. P. Plaks and P. M. Athanas, Eds. CSREA Press, June 2002, pp. 49–53.

[3] A. S. Dawood, S. J. Visser, and J. A. Williams, "Reconfigurable FPGAs for Real Time Image Processing in Space," in *14th International Conference on Digital Signal Processing (DSP 2002)*, vol. 2, 2002, pp. 711–717.

[4] J. Villasenor and B. Hutchings, "The flexibility of configurable computing: Providing the hardware for data-intensive real-time processing," pp. 67–84, Sept. 1998.

[5] B. Pratt, M. Caffrey, P. Graham, K. Morgan, and M. Wirthlin, "Improving FPGA design robustness with partial TMR," in *44th Annual IEEE International Reliability Physics Symposium Proceedings*, 2006, pp. 226–232.

[6] Xilinx, "Radiation hardened Virtex-II QPRO 1.5V platform FPGAs: Introduction and overview," Xilinx, Inc., San Jose, CA, Datasheet DS124-1, July 2003.

[7] C. Carmichael, M. Caffrey, and A. Salazar, "Correcting single-event upsets through Virtex partial configuration," *Xilinx Application Notes, XAPP216 (v1. 0)*, 2000.

[8] F. Lima, C. Carmichael, J. Fabula, R. Padovani, R. Reis, X. Inc, and C. San Jose, "A fault injection analysis of Virtex FPGA TMR design methodology," in *Radiation and Its Effects on Components and Systems, 2001. 6th European Conference on*, 2001, pp. 275–282.

[9] B. Bridgford, C. Carmichael, and C. W. Tseng, "Single-event upset mitigation selection guide," *Xilinx Application Note XAPP987*, vol. 1, 2008.

[10] K. S. Morgan, "SEU-Induced Persistent Error Propagation in FPGAs," Master's thesis, Brigham Young University, August 2006.

[11] D. Siewiorek and R. Swarz, *Reliable computer systems: design and evaluation*. AK Peters, Ltd.

[12] D. McMurtrey, K. Morgan, B. Pratt, and M. Wirthlin, "Estimating TMR reliability on FPGAs using markov models." [Online]. Available: http://hdl.handle.net/1877/644

[13] K. Morgan, M. Caffrey, P. Graham, E. Johnson, B. Pratt, and M. Wirthlin, "SEU-induced persistent error propagation in FPGAs," *IEEE Transactions on Nuclear Science*, vol. 52, no. 6 Part 1, pp. 2438–2445, 2005.

[14] R. Karp, *Reductibility among combinatorial problems*. Univ. of California, 1972.

[15] G. Even, "Approximating minimum feedback sets and multicuts in directed graphs," *Algorithmica*, vol. 20, no. 2, pp. 151–174, 1998.