OVERHEAD AND RELIABILITY ANALYSIS OF ALGORITHM-BASED FAULT TOLERANCE IN FPGA SYSTEMS

Adam Jacobs, Grzegorz Cieslewski, and Alan D. George

NSF Center for High-Performance Reconfigurable Computing (CHREC)
Department of Electrical and Computer Engineering, University of Florida
email: {jacobs, cieslewski, george}@chrec.org

ABSTRACT

Commercial SRAM-based, field-programmable gate arrays (FPGAs) have the capability to provide space applications with the necessary performance, energy-efficiency, and adaptability to meet next-generation mission requirements. However, mitigating an FPGA's susceptibility to radiation-induced faults is challenging. Triple-modular redundancy (TMR) techniques are traditionally used to mitigate radiation effects, but TMR incurs substantial overheads such as increased area and power requirements. In order to reduce these overheads while still providing sufficient radiation mitigation, we propose the use of algorithm-based fault tolerance (ABFT). We investigate the effectiveness of hardware-based ABFT logic in COTS FPGAs by developing multiple ABFT-enabled matrix multiplication designs, carefully analyzing resource usage and reliability tradeoffs, and proposing design modifications for higher reliability. We perform fault-injection testing on a Xilinx Virtex-5 platform to validate these ABFT designs, measure design vulnerability, and compare ABFT effectiveness to other fault-tolerance methods. Our hybrid ABFT design reduces total design vulnerability by 99% while only incurring 25% overhead over a baseline, non-protected design.

1. INTRODUCTION

As remote sensor technology for space systems increases in fidelity, the amount of data collected by orbiting satellites and other space vehicles will continue to outpace the ability to transmit that data to other stations (e.g., ground stations, other satellites). By increasing the onboard data-processing capabilities of future systems, raw data can be interpreted, reduced, and/or compressed onboard the space system before transmitting the results to ground stations, thus reducing data transmission requirements. Traditionally, radiation-hardened devices, with increased protection from long-term radiation exposure (total ionizing dose), provide system reliability and correctness for these space systems. However, these hardened devices are very expensive and have dramatically reduced performance as compared to non-hardened commercial-off-the-shelf (COTS) components.

One approach for high-performance space system design leverages hardware-adaptive devices such as field-programmable gate arrays (FPGAs). COTS FPGAs can provide parallel computations at a high level of performance per unit size, mass, and power [1]. Fortunately, many space applications, such as synthetic aperture radar (SAR) [2], hyperspectral imaging (HSI) [3], image compression [4], and other image processing applications [5], where onboard data processing can significantly reduce data transmission requirements, are amenable to an FPGA's highly parallel architecture.

In order to leverage FPGAs in space systems, the FPGA must operate correctly and reliably in high-radiation environments. Fortunately, when combined with special system design techniques, SRAM-based FPGAs can be viable for space systems. An SRAM-based FPGA's primary computational limitation is the possibility of SEUs causing errors within the FPGA user logic and routing resources, which can manifest as configuration memory upsets or logic memory (e.g., flip-flops, user RAM) upsets, resulting in deviations from the expected application behavior. Fault-tolerant techniques, such as triple-modular redundancy (TMR) and memory scrubbing, can protect the system from most SEUs and significantly decrease the SEU-induced errors, but designing an FPGA-based space system using TMR introduces at least 200% area overhead for each protected module. Depending on the expected upset rates for a given space system, other fault-tolerance methods could be used to provide sufficient reliability while maximizing the resources available for performance.

Algorithm-based fault tolerance (ABFT) is a method that can be used with many linear-algebra operations, such as matrix multiplication or LU decomposition [6]. Fortunately, many space applications are composed of linear-algebra operations; e.g., HSI features matrix multiplication, while SAR features fast Fourier transforms. Other algo-

This work was supported in part by the I/UCRC Program of the National Science Foundation under Grant No. EEC-0642422.

rithms can often be converted to fit an algebraic framework. Traditionally, ABFT has been implemented in software, with multiprocessor arrays, and in hardware, with systolic arrays, to protect application datapaths. Our ABFT approach may be used in FPGA applications to provide both datapath and configuration memory protection with low overhead.

We present an analysis of multiple fault-tolerance methods on Xilinx FPGAs including TMR and ABFT. We examine the resource usage of each method and measure the vulnerability of the design using a fault-injection tool. We then examine possible design tradeoffs and modifications that can enable higher reliability. The remaining sections of this paper are organized as follows. Section 2 surveys previous work related to ABFT. Section 3 describes multiple hardware architectures for a matrix multiplication design that is used as an example case study. Section 4 analyzes resource usage and vulnerability of each design using a fault-injection tool. Finally, Section 5 presents conclusions and outlines directions for future research.

2. BACKGROUND

ABFT augments an original data matrix with row and/or column checksums and the linear-algebra operation is performed on the new, augmented matrix. If the linear-algebra operation is computed successfully, the resulting augmented matrix will contain valid, consistent checksums [6]. ABFT checksum generation and comparison has lower computational complexity than the primary linear-algebra operation. ABFT computational overhead is generally low, and as a proportion of total computation, decreases as the matrix size increases. ABFT was originally envisioned for use with massively parallel processor and systolic arrays, but can also be created within an FPGA's logic fabric. Our work shows that ABFT in FPGAs can be used to detect many errors caused by configuration memory faults in addition to data memory faults.

2.1. Mathematical Basis for ABFT

The following definitions provide the mathematical background for ABFT. To obtain the weighted checksums, the initial data will have to be multiplied by an encoder matrix. Without loss of generality and to simplify the notation, we assume that generic matrix is square with dimensions of $N \times N$.

Definition 1: An encoder matrix is a matrix whose product with the data matrix will yield the desired checksums. For the remainder of this paper we will refer to the encoder matrix as E_N. The E_N used in this paper will have dimensions of $N \times 1$.

$$E_N = \left[\begin{array}{ccccc} 1 & 1 & \cdots & 1 & 1 \end{array} \right]^T \tag{1}$$

Definition 2: A column checksum matrix A_C is an initial data matrix A that has been augmented with extra rows of checksums. Such a matrix will have dimensions of $(N + 1) \times N$ and has the form:

$$A_C = \left[\begin{array}{c} A \\ E_N^T \cdot A \end{array} \right] \tag{2}$$

Similarly, a row checksum matrix A_R can be obtained by augmenting a data matrix B with additional columns of the following form:

$$A_R = \left[\begin{array}{cc} A & A \cdot E_N \end{array} \right] \tag{3}$$

Definition 3: The product of a column checksum matrix A and a row checksum matrix B will produce a full checksum matrix C_F. Such a matrix will have dimensions of $(N + 1) \times (N + 1)$ and the form:

$$\begin{aligned} A_C \cdot B_R &= \left[\begin{array}{c} A \\ E_N^T \cdot A \end{array} \right] \cdot \left[\begin{array}{cc} B & B \cdot E_N \end{array} \right] \\ &= \left[\begin{array}{cc} A \cdot B & A \cdot B \cdot E_N \\ E_N^T \cdot A \cdot B & E_N^T \cdot A \cdot B \cdot E_N \end{array} \right] \\ &= \left[\begin{array}{cc} C & C \cdot E_N \\ E_N^T \cdot C & E_N^T \cdot C \cdot E_N \end{array} \right] \\ &= C_F \end{aligned} \tag{4}$$

The associative property of the matrix product allows for verification of the multiplication procedure by simply recalculating the checksums and comparing them with ones obtained through the matrix multiplication. In general, operations that preserve weighted checksums are called checksum-preserving and the matrix product is an example of such a function.

The ABFT method has also been expanded from matrix multiplication to protect other algorithms comprised of linear operations, such as QR decomposition or the Fast Fourier Transform [7] [8]. Silva et al. investigated vulnerabilities in traditional ABFT implementations and proposed methods for improving fault coverage using a Robust ABFT approach [9].

2.2. Alternative Fault-Tolerance Techniques

Traditionally, the primary methods for providing fault tolerance to FPGA systems have been TMR and configuration scrubbing. TMR masks any single fault through majority voting, and configuration scrubbing repairs faults to prevent accumulation of multiple errors. Other fault-tolerance strategies focus on similar redundancy-based techniques. Duplication with compare (DWC) replicates an application module and uses a simple comparison operator to detect mismatches in output, requiring reconfiguration and recomputation to correct the error [10]. Shim et al. used a

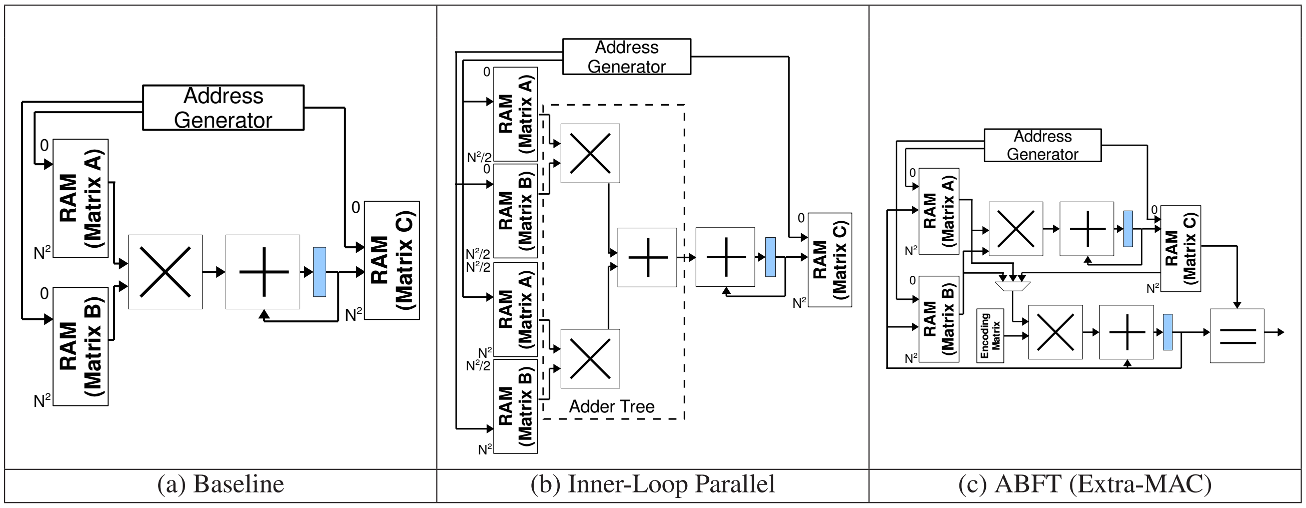

Fig. 1. Matrix Multiplication Architectures

technique known as reduced-precision redundancy (RPR) to reduce overhead in systems which can tolerate small levels of noise [11]. RPR triplicates an application module as in TMR, but the replicas have lower precision or only operate on the most-significant bits of application data, ensuring that the most-significant bits are protected and errors in the least-significant bits are treated as noise in the system. For applications that are not comprised of linear-algebra operations, and therefore cannot be protected using ABFT, these other fault-tolerance techniques can be used to increase system reliability.

3. FPGA-BASED MATRIX MULTIPLICATION

Matrix multiplication (MM) is used as a key kernel in a large number of signal-processing applications, and can benefit from the performance of FPGAs. A matrix multiplication module was created to examine the reliability of hardware-based ABFT. For this analysis, 32-bit integer precision was used in each design. This section discusses one possible parallelization strategy for MM and the design decisions that were made for the ABFT architecture.

3.1. Baseline, Serial Architecture

The minimal-hardware, serial architecture for the matrix multiplication function consists of a single multiply-accumulator (MAC), an address generation module, and three data storage modules (RAM) as shown in Fig. 1(a). Two memories are used to store the input matrices A and B, and one memory is used for the resulting output matrix C. The address generator iterates through the correct matrix indices, sending data stored in the two input RAMs to the MAC, and generates the appropriate address for output values. This MM module can be used for any size matrix, the limiting factor being data storage. This architecture requires N^3 cycles to fully calculate an output matrix. MM computational throughput can be improved by exploiting parallelism with additional MAC units.

3.2. Inner-Loop Parallel Architecture

The inner-loop parallel MM architecture unrolls the inner loop of the MM algorithm as shown in Fig. 1(b). Each element in the result matrix C is the dot product of a row from matrix A and a column from matrix B. This parallel architecture uses multiple processing elements to compute the dot-product in parallel. With the fine-grained parallel architecture shown in Figure 1(b), the output of several multipliers are connected to an adder-tree structure, allowing the parallel computation of partial dot-products, which are then accumulated into the final, full dot product. By fully parallelizing the dot product (using N multipliers), the execution time of the full algorithm can be reduced from $O(N^3)$ to $O(N^2)$. This method requires accessing multiple memory elements in parallel, and may be limited by the total number of memory elements on the FPGA.

3.3. ABFT Hardware Requirements

The addition of ABFT logic requires the creation of two functions, ABFT checksum generation and ABFT checksum validation. Each of these functions requires a simple accumulator (or a MAC for weighted checksums). Checksum generation sums each column of matrix A and writes the checksum into the matrix A RAM. Next, checksum generation does the same process for the rows of matrix B. The checksum validation function sums the columns of matrix C and compares the sum to the checksum value in the matrix C RAM. If a mismatch is detected, an "error

Table 1. Resource Utilization and Overhead of MM Designs

Design Name	Look-up Tables (LUT)	LUT Overhead	Flip-Flops (FF)	FF Overhead	BlockRAM (BRAM)	BRAM Overhead	DSP48	DSP48 Overhead
Baseline (logic)	3,431	–	1,039	–	48	–	0	–
Baseline (DSP48)	1,476	–	1,017	–	48	–	12	–
ABFT (shared MAC)	1,849	25%	1,119	10%	48	0%	12	0%
ABFT (extra MAC)	1,781	21%	1,265	24%	48	0%	15	25%
TMR	3,658	148%	1,873	84%	144	200%	36	200%
Hybrid ABFT	3,869	162%	1,465	44%	48	0%	15	25%

found" signal is asserted until the module is reset. For some applications, such as image processing, data errors that occur in low-significance bits may be ignored. ABFT accomplishes this by comparing the difference of two generated checksums to a user-defined threshold value. For maximum coverage, this threshold should be set to zero for integer operations. Figure 1(c) shows an example of an ABFT-enable MM architecture where an additional MAC is used for the checksum generation and validation functions. The MAC hardware that exists for the main MM operation may be reused for creating checksums (shared-MAC), or an additional accumulator can be used for this purpose (extra-MAC). For the baseline architecture, an additional ABFT accumulator would incur almost 100% overhead. However, for parallel designs with multiple processing elements, the overhead is amortized.

To implement ABFT error correction, the column and row indices of faulty rows can be temporarily stored in registers. The checksum generation module would then recalculate the column checksum, ignoring the value at the faulty row index. This sum would be subtracted from the matrix C checksum value to obtain the correct value, and stored in the matrix C RAM. Multiple errors can be detected, but the correction algorithm may fail, depending on the encoding matrix used for ABFT. The ABFT designs discussed in Section 4 perform error detection only.

4. RESOURCE-OVERHEAD EXPERIMENTS

In this section we analyze the overhead of the architectures presented in Section 3 and compare them to traditional fault-tolerance mitigation strategies. For this analysis, we use a 32-bit integer MM module with 4 processing elements. This MM module can perform computation on matrices up to 128×128 elements in size. This module is also the largest module that will fit in the Xilinx V5LX110 FPGA when using TMR. We compare a baseline inner-loop MM design with several fault-tolerant designs (TMR, ABFT, and hybrid TMR/ABFT). The results of this comparison are shown in Table 1.

4.1. Hardware ABFT Resource Overhead

The baseline architecture uses 48 BlockRAMs to store input and output data. When the design is implemented completely in the FPGA logic fabric, 3,431 LUTs and 1,039 FFs are required. However, by changing synthesis options in the Xilinx ISE tool, the multiply-accumulators can be implemented in 12 DSP48 units (3 per 32-bit MAC), reducing the needed LUTs by almost 2,000. The remaining LUTs and FFs are used to create the counters and state machines within the address generation unit.

The shared-MAC ABFT design does not require any additional BRAM or DSP48 units over the baseline design. The additional logic needed to handle addressing matrices during checksum generation and validation increase the required LUTs by 25% and FFs by 10%. The extra-MAC ABFT design uses more DSP48s and FFs than the shared-MAC design, while using fewer LUTs due to removed data multiplexers.

For comparison, Table 1 also shows the resource usage of a TMR design. This TMR design was created by replicating the entire ABFT-MM core in the top-level VHDL code and creating a majority voter for each of the outputs. Alternative methods for creating TMR designs are available from Xilinx [12], BYU-LANL [13], Mentor Graphics [14], and others. As expected, 200% more BRAMs and DSPs are required for TMR. However, the LUT and FF usage did not increase as much as expected (but still much more than in the ABFT designs). This difference may be caused by optimization during the Xilinx synthesis or place-and-route processes. We also examine a hybrid design based on the extra-MAC design which uses TMR on the address generator and all state machines within the design, but only uses ABFT along the data path. This hybrid approach results in a design that has approximately 162% overhead for LUTs but only 25% overhead on the limited DSP48 resources.

4.2. Hardware ABFT Scalability

As more processing elements are used in the MM module, the overhead created by the additional ABFT MAC unit becomes extremely low. Additionally, the ABFT method

Table 2. Matrix Multiplication Fault-Injection Results

Design Name	Faults Injected	Data Errors	System Hangs	Vulnerable Config Bits (est.)	Vulnerable RAM Bits	DVF (%)
Baseline (logic)	100,000	1,216	68	60,376	1,572,864	10.36%
Baseline (DSP48)	100,000	719	14	34,467	1,572,864	10.20%
ABFT (shared MAC)	100,000	351	67	17,111	0	0.14%
ABFT (extra MAC)	100,000	351	40	16,005	0	0.13%
TMR	100,000	42	12	5,158	0	0.041%
Hybrid ABFT	100,000	261	19	11,461	0	0.092%

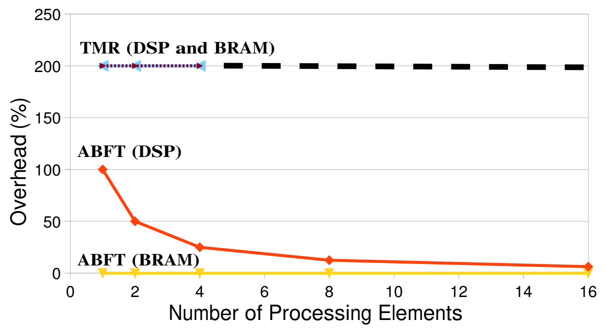

Fig. 2. Overhead of Parallel Matrix Multiplication

does not require additional BRAMs. For modules with a large number of processing elements, the control logic can be significantly more complicated than the baseline, unprotected design. However, the MM design uses a very small portion of the logic available on the FPGA, and the design is limited only by BRAM and DSP48 availability.

Figure 2 compares the overhead of an ABFT design to a TMR design while varying the number of processing elements. For highly parallel designs, the ABFT extra-MAC architecture has 6.25% DSP overhead with 0% BRAM overhead. Meanwhile, the TMR design uses 200% more of the limited BRAM and DSP FPGA resources. The TMR design is limited to only 4 processing elements, while the ABFT designs are able to scale up to 16 processing elements before running out of DSP resources.

5. FAULT-INJECTION EXPERIMENTS

While Section 4 has shown that ABFT provides lower overhead compared to other fault-tolerance strategies, the reliability of ABFT must also be evaluated. In order to validate our ABFT design, faults must be injected into an executing system. In this section, we present FPGA fault-injection results gathered using the Simple, Portable Fault Injector (SPFI) [15]. SPFI performs fault injection by modifying configuration frames within a design's bitstream, re-programming the FPGA, and comparing the resulting output against known values. Partial reconfiguration is used to reduce the time required to modify configuration memory and to improve the speed of fault injection. However, due to the length of time required to exhaustively test an entire FPGA design, statistical sampling is used to estimate the total number of vulnerable bits in a given design.

We implemented multiple fault-tolerant designs discussed in Section 3 on a Xilinx ML505 FPGA development platform. Each design used a 32-bit integer MM module with 4 processing elements. A UART was also implemented on the FPGA to stream input test vectors to the MM module and to report results back to a verification program. During the design phase, the MM module is constrained to a specified portion of the FPGA. The SPFI fault-injection tool enables targeted injections, allowing the UART to be avoided during fault injection. Table 2 shows the fault-injection results for multiple fault-tolerant MM designs. For each design, the table indicates the number of injections performed and the number of data errors and system hangs detected. In this analysis, bits resulting in false-positive ABFT results were not considered vulnerable, since they indicate an error in the validation logic being detected. The measured percentage of faults is then scaled to the size of the injection area to estimate the total number of vulnerable bits. The fault vulnerability for each component is scaled to the FPGA's total number of configuration bits to estimate the components' design vulnerability factor (DVF). A design's DVF represents the percentage of bits that are vulnerable to faults and can result in errors. Reliability of a given design can then be calculated from the FPGA's total fault rate scaled by the design's DVF. Most Xilinx FPGA designs have a DVF that ranges from 1% to 10% [16] due to the large amount of configuration memory devoted to routing.

The baseline MM design has approximately 60,376 vulnerable configuration bits. Additionally, the data stored in BlockRAM is unprotected and can result in incorrect calculations. The baseline MM design with DSP48 units has a significantly lower amount of vulnerable configuration bits. Using the built-in DSP48 units increases reliability of the design because a DSP48 multiplier uses fewer configuration bits than a multiplier implemented in the FPGA logic fabric.

Implementing addressing logic with DSP48 may enable an even more reliable design.

The ABFT shared-MAC design has an estimated 17,111 vulnerable bits (about 50% of the unprotected design). All of the BlockRAM data bits are protected through the ABFT algorithm, resulting in a significantly lowered total DVF. The vulnerable bits can affect address generation, the checksum generation or validation, or the error detection status register. By using an independent MAC unit for checksum generation and verification, faults in the main data-path do not affect checksum calculations, leading to a more reliable design. As shown, the extra-MAC design has fewer vulnerable bits than the shared-MAC design. Both ABFT designs experience a similar amount of data errors, but fewer faults cause the extra-MAC design to hang. An examination of result matrices with data errors reveals that many faulty matrices contained all zero values, producing an incorrect result with a valid (zero) checksum. It may be possible to further improve the reliability of ABFT designs by modifying the ABFT encoding matrix to ensure that output matrices cannot contain all zeros.

The TMR design has an estimated 5,158 vulnerable bits and a DVF of 0.041%. The majority of these bits are the result of routing faults or TMR majority voter faults. This result represents a realistic lower bound on total vulnerability of the MM design. However, using third-party TMR design tools may result in higher-reliability designs than our high-level TMR approach, since these tools can perform TMR with finer granularity and more frequent voting.

The hybrid ABFT design has approximately 30% lower vulnerability than the extra-MAC design, but higher vulnerability than the TMR design. However, the hybrid design has lower resource usage compared to the TMR design (see Table 1). For applications where BRAM or DSP48 resources are used extensively, the hybrid ABFT design provides a good compromise between low vulnerability (0.09% DVF) and low DSP48 usage (25% overhead).

6. CONCLUSIONS

In this work, we have presented a novel analysis of ABFT for low-overhead fault tolerance in FPGA systems. Several matrix multiplication designs employing TMR and ABFT fault-tolerance techniques were developed and tested using a FPGA fault-injection tool. The results showed that ABFT was capable of reducing the number of vulnerable configuration bits in a design while also protecting all memory bits. While the TMR fault-mitigation approach had the lowest vulnerability, a hybrid ABFT/TMR design was able to balance low design vulnerability (0.09%) with low resource overhead (25%).

Future work will examine design techniques to further improve the reliability of ABFT designs on FPGAs, such as modifying the ABFT encoding matrix. We will also evaluate the effectiveness of ABFT on other algorithms such as FFT and QR decomposition.

7. REFERENCES

[1] J. Williams, C. Massie, A. D. George, J. Richardson, K. Gosrani, and H. Lam, "Characterization of fixed and reconfigurable multi-core devices for application acceleration," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 3, pp. 19:1–19:29, November 2010. [Online]. Available: http://doi.acm.org/10.1145/1862648.1862649

[2] C. Le, S. Chan, F. Cheng, W. Fang, M. Fischman, S. Hensley, R. Johnson, M. Jourdan, M. Marina, B. Parham, F. Rogez, P. Rosen, B. Shah, and S. Taft, ""onboard fpga-based sar processing for future spaceborne systems"," in *Proceedings of the IEEE Radar Conference, 2004.*, april 2004, pp. 15 – 20.

[3] M. Hsueh and C.-I. Chang, ""field programmable gate arrays (fpga) for pixel purity index using blocks of skewers for endmember extraction in hyperspectral imagery"," *Int. J. High Perform. Comput. Appl.*, vol. 22, pp. 408–423, November 2008. [Online]. Available: http://portal.acm.org/citation.cfm?id=1453081.1453083

[4] A. Gupta, S. Nooshabadi, D. Taubman, and M. Dyer, "Realizing low-cost high-throughput general-purpose block encoder for jpeg2000," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 16, no. 7, pp. 843 –858, july 2006.

[5] A. Dawood, S. Visser, and J. Williams, "Reconfigurable fpgas for real time image processing in space," in *14th International Conference on Digital Signal Processing, 2002. DSP 2002.*, vol. 2, 2002, pp. 845 – 848 vol.2.

[6] K.-H. Huang and J. Abraham, "Algorithm-based fault tolerance for matrix operations," *IEEE Transactions on Computers*, vol. C-33, no. 6, pp. 518 –528, june 1984.

[7] S.-J. Wang and N. Jha, "Algorithm-based fault tolerance for fft networks," *IEEE Transactions on Computers*, vol. 43, no. 7, pp. 849 –854, jul 1994.

[8] G. Cieslewski, A. Jacobs, and A. George, "Fault-tolerant 2d fourier transform with checksum encoding," in *Aerospace Conference, 2007 IEEE*, march 2007, pp. 1 –11.

[9] J. Silva, P. Prata, M. Rela, and H. Madeira, "Practical issues in the use of abft and a new failure model," in *Twenty-Eighth Annual International Symposium on Fault-Tolerant Computing*, jun 1998, pp. 26 –35.

[10] J. Johnson, W. Howes, M. Wirthlin, D. McMurtrey, M. Caffrey, P. Graham, and K. Morgan, "Using duplication with compare for on-line error detection in fpga-based designs," in *2008 IEEE Aerospace Conference*, March 2008, pp. 1–11.

[11] B. Shim, S. Sridhara, and N. Shanbhag, “Reliable low-power digital signal processing via reduced precision redundancy,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 5, pp. 497 – 510, May 2004.

[12] *XTMR Tool User Guide*, Xilinx, 2004, xilinx User Guide UG156.

[13] 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.*, march 2006, pp. 226 –232.

[14] *Precision Hi-Rel Datasheet*, Mentor Graphics, 2011, http://www.mentor.com/products/fpga/synthesis/precision-hi-rel/.

[15] G. Cieslewski, A. George, and A. Jacobs, “Acceleration of fpga fault injection through multi-bit testing,” in *2010 Engineering of Reconfigurable Systems and Algorithms*, July 2010.

[16] *SEU Strategies for Virtex-5 Devices*, Xilinx, 2010, xilinx Application Note XAPP864.