Defense Threat Reduction Agency/J9NT
8725 John J. Kingman Road, MSC 6201
Fort Belvoir, VA 22060-6201

FINAL REPORT

Modeling and Simulation of Error Code-based SEU Radiation-Hardening for Computational and Boolean Logic Elements (CORRECT)

University of Southern California, Information Sciences Institute

Contract no. HDTRA1-12-C-0057
Contract Data Requirement List –Data Item A003

Period of Performance 05/17/2012 – 11/17/2013

Technical POC	Administrative POC
Michael Fritze USC Information Sciences Institute 3811 N. Fairfax Drive, Suite 200 Arlington, VA 22203 Email: mfritze@isi.edu Tel: 703-812-3715	Barbara Reguengo USC Information Sciences Institute 4676 Admiralty Way Marina del Rey, Suite 1001 Marina Del Rey, CA 90292 Email: **reguengo@usc.edu** Tel: 310-448-8412

DTRA Project Manger: Les Palkuti (leslie.paltuki@dtra.mil)
Tel: (703)-767-7455

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
05-30-2014	Final Report	17-05-2012 to 17-11-2013

4. TITLE AND SUBTITLE
Modeling and Simulation of Error-Code-based SEU Radiation-Hardening for Computational and Boolean Logic Elements (CORRECT)

5a. CONTRACT NUMBER
HDTRA1-12-C-0057

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Michel Sika, Jonathan Ahlbin, Michael Bajura, Michael Fritze

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University Of Southern California, Information Sciences Institute
University Gardens, Ste. 203
Los Angeles, CA 90089-0001

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Threat Reduction Agency (DTRA) / J9NT
8727 John J Kingman Road
Fort Belvoir, VA 22060-6201

10. SPONSOR/MONITOR'S ACRONYM(S)
DTRA

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
Distribution Statement A
"Approved for Public Release; Distribution is Unlimited"

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The main objective of the CORRECT program was to investigate the applicability of Residue Arithmetic Coding (RAC) to mitigate single-event upsets in the arithmetic and boolean processing components of computing logic chains. The potential impact of this work is an alternative approach to hardening with much lower overhead compared with conventional methods (e.g. triple-modular redundancy). Over the 18 months of the program, an algorithm trade-off study and the design of basic residue hardened basic logic elements were successfully performed.

15. SUBJECT TERMS
Radiation Effects in Semiconductors; Error Detection and Correction; Single-Event Upset; Computational Architectures

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE	None	33 incl. cover + attach	Michael Bajura
Unclassified	Unclassified	Unclassified			19b. TELEPHONE NUMBER *(include area code)* 703-812-3719; mbajura@isi.edu

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

Definitions	
ASIC	**Application Specific Integrated Circuit**
CAD	**Computer-Aided Design**
CORRECT	**Modeling & Simulation of Error Code-based SEU Radiation-Hardening for Computational and Boolean Logic Elements**
DARPA	**Defense Advanced Research Projects Agency**
ED	**Error Detection**
EDAC	**Error-Detection And Correction**
LEAP	**Leading-Edge Access Program**
RAC	**Residue Arithmetic Coding**
RHBD	**Radiation Hardening By Design**
SOI	**Silicon-On-Insulator**
SET	**Single-Event Transient**
SEU	**Single-Event Upset**
TAPO	**Trusted Access Programs Office**
TMR	**Triple-Modular Redundancy**

I. EXECUTIVE SUMMARY

The main objective of the CORRECT program has been investigating the applicability of Residue Arithmetic Coding (RAC) to mitigate single-event upsets in the arithmetic and Boolean processing components of computing logic chains. The potential impact of this work is an alternative approach to hardening with much lower overhead compared with conventional methods (e.g. triple-modular redundancy). Over the 18 months of the program, an algorithm trade-off study was performed and basic residue hardened basic logic elements were designed. Basic RAC-hardened computational logic blocks were also designed and fabricated in ASICs to be evaluated in a follow-on phase. The program was categorized into three sections: library development, architecture evaluation, and fault injection framework with multiple milestones and deliverables for each category as illustrated in Figure 1.

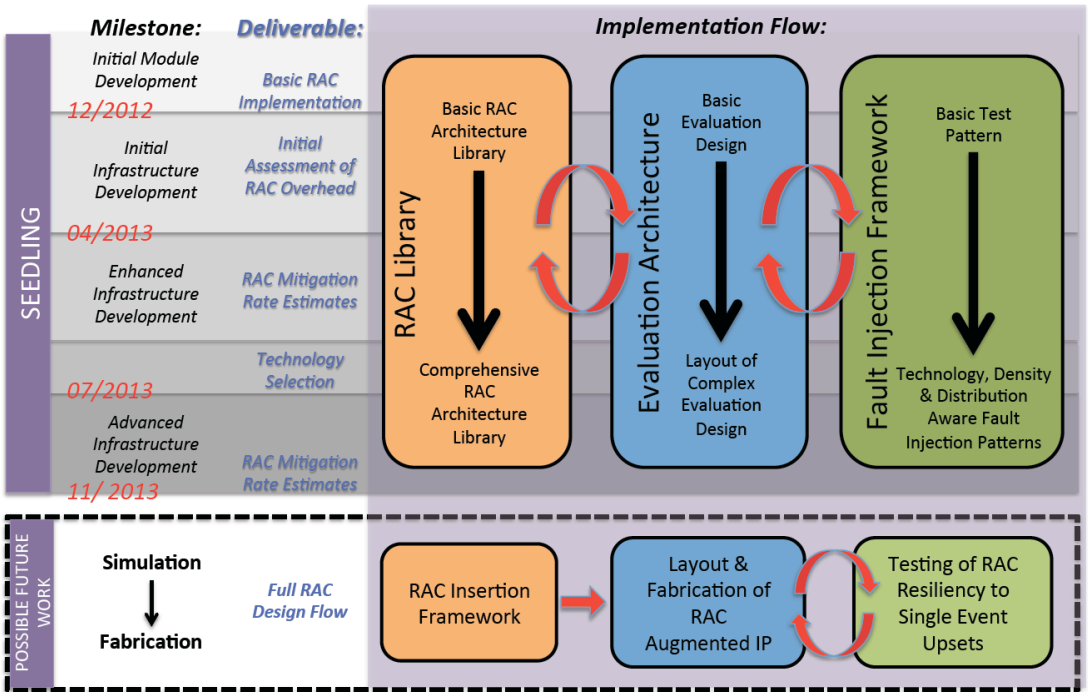

Figure 1 CORRECT 18-month execution plan.

Within the library development section, 8, 16, 32, 64, and 128-bit versions of adders and multipliers were created. These basic elements were then evaluated within the architecture and fault injection framework sections of the project. Various designs and residue hardening schemes were devised and tested with in-house fault injection tools and commercial simulation and CAD tools. For example for the 45 nm SOI 8-bit full adder, simulation analysis showed that residue hardening of the adder detected 99.83% of $1x10^6$ single upsets simulated. With the success of the simulation study, the 8-bit adder was designed in the IBM 12SOI process technology and sent for fabrication through the DARPA – TAPO LEAP program. Fabricated test chips will allow for verification of the simulation work and for continued work in this area.

II. ACCOMPLISHMENTS

A brief description of recent accomplishments in the CORRECT program is given here whereas a comprehensive technical description of the test structures, experimental setup and procedures and data analysis is provided in section III.

A. Error-resilient Algorithms and the Architecture Trade-offs for Basic Logic Elements

The Residue Arithmetic Coding or RAC is based on the detection of faults concurrently to the main operation using operations on residue-coded operands. With RAC, error detection does not add delay to the computation of the main operation. Correction is achieved with minor additional delay by evaluating the outcome of the operations on the residue-coded operands against the residue-coded result of the computation component undergoing hardening. The ability of residue codes to perform concurrent operations on residue-coded operands is what presents RAC with the ability to detect and correct with much lower overhead than redundancy based RHBD techniques and less delay than temporal filtering techniques. RAC for error detection (ED) or error detection and correction (EDAC) are computations on the residues of the set of operands R for a given modulus or set of moduli M.

A residue is the remainder from the operation of dividing an integer by a modulus. The residue of an n-bit operand A modulo m labeled $[A]_m$, is the remainder obtained from dividing A by m. If R is the set of operands where $R=\{A, B,...\}$ and M the set of moduli where $M=\{m_0, m_1,...\}$ then R_M is the set of resulting residues $R_M = \{\{[A]_{m0}, [A]_{m1} ...\} ,\{[B]_{m0}, [b]_{m1} ...\}, ...\}$ where each operand is reduced modulo each modulus. If the compute kernel has inputs in the form of operands R and yields an output labeled S, then S_M is the compute kernel output reduced modulo M. The sets of M and R_M are used to uniquely characterize error patterns in an arithmetic operation [3][4]. R_M is compared with the residue set S_M in order to detect and correct SEU errors. The number and arithmetic properties of the selected M determines the "arithmetic weight" of the errors that RAC can detect and/or correct, and consequently the EDAC "strength" of the code. Although a single modulus is appropriate for error detection, error correction with a single modulus is not reliable unless the modulus does not result in congruencies when comparing R_M and S_M. Such a modulus however would need to be significantly larger in terms of number of bits compared to the size of the output of the compute kernel that is to be hardened. For reduced overhead and improved efficiency in error correction, an additional modulus is used for every bit of correction that is required in lieu of a single large modulus. Single-bit error correction requires the selection of two moduli, dual-bit error correction requires the selection of three moduli, et cetera. This work evaluates the effectiveness of two-moduli based RAC designed for single-bit correction and multi-bit detection, this is known as a bi-residue code.

Figure 2 schematically presents the application of residue coding to the hardening of an N-bit ADDER for single-bit detection and correction capability [4].

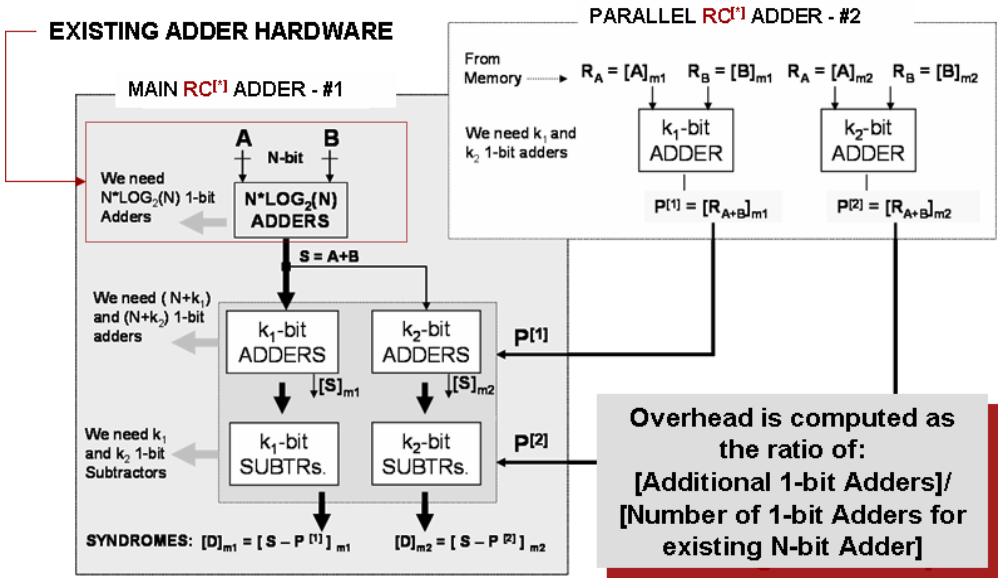

Figure 2 Single-bit ED and EDAC applied to and N-bit adder using two moduli.

Example - RAC error detection:

For a given modulus ***m*** and a simple two-input single-output operation ***OP*** with operands ***A***, ***B***, and result ***S***, we compute:

(1)	$S = A$ ***OP*** B	*; Perform operation **OP** on the Operands*
(2)	$[S_1]_m = S \bmod m$	*; Compute the residue of the result reduced* *; modulo m*
(3)	$[A]_m = A \bmod m$ $[B]_m = B \bmod m$	*; Compute The residue of the operands reduced* *; modulo m*
(4)	$[S_2]_m = ([A]_m$ ***OP*** $[B]_m) \bmod m$	*; Perform operation OP on the Operand* *; residues reduced modulo m*
(5)	$[S_1]_m$ *vs* $[S_2]_m$	*; Compare the residue of the operation OP* *; applied to the Operands against the operation* *; OP applied to the residues of the Operands*

An error is detected when the residues in (5) do not match:

IF $[S_1]_m \neq [S_2]_m$ THEN ERROR

If there is a single-bit error error in the original operation, the difference between the moduli of the result can be expressed in terms of the residue of the incorrect bit:

$$[S_1]_m = [S_2]_m \pm \text{OP}\ [2^i]_m$$

Where *i* indicates the position of the error reflected in the output of the operation. The example shows that it is possible to detect a single error with a single modulus. With a single modulus however, it is typically not practical to reliably find and correct the erroneous bit error, as two or more *i* can be congruent modulo ***m***. If however we introduce a second modulus ***n*** and designate ***r*** and ***s*** as two *i* such that:

$$(1)\ r \equiv s \bmod m$$

then (1) implies that we can not also have:

$$r \equiv s \bmod n$$

if ***n*** and ***m*** that are relatively prime and each with magnitude in excess of the maximum range for ***r*** and ***s*** [5].

To illustrate the above computations numerically – consider an operation of two-input *addition* with operands 9 and 13, using a modulus of 7:

(1)	$S = 9 + 13 = \cancel{22} = 23$	*; Addition with single bit error (23 instead of 22)*
(2)	$[S_1]_7 = 23 \bmod 7 = 2$	*; Compute the residue of the result reduced ; modulo m*
(3)	$[9]_7 = 9 \bmod 7 = 2$ $[13]_7 = 13 \bmod 7 = 6$	*; Compute The residue of the operands 9 and 7 ; reduced modulo 7*
(4)	$[S_2]_7 = ([9]_7 + [13]_7) \bmod 7$ $= (2 + 6) \bmod 7$ $= 8 \bmod 7 = 1$	*; Perform addition on the Operand residues ; reduced modulo 7*
(5)	$[S_1]_m$ *vs* $[S_2]_m$ $2 \neq 1$	*; The residue of the sum of 9 and 13 reduced modulo 7 against the sum of the residues of 9 and 13 reduced modulo 7*

Here **A**=9, **B**=13, and **m**=7. Thus **S** = 22, $[S_1]_m$ = 1, $[A]_m$ = 2, $[B]_m$ = 6, $[S_2]_m$ = 1. If there is a single-bit error in the computation – e.g. the lowest order bit is flipped in the result (making it 23 instead of 22): Then $[S_1]_m$ = 2, and $[S_2]_m$ = 1. Analogously, a single-bit error in the residue computation logic would yield and incorrect $[S_2]_m$ instead of an incorrect $[S_1]_m$ value. The combination of the single-bit error example in the main operation and the analogous fault in the error detection logic illustrates that RAC logic does not require protection.

With the above conditions we can find the position of a single bit error with two moduli. The residue sets of the operands and the computation result are compared pairwise for each modulus. If both pairs of residue differences do not match, error correction is implemented by flipping the appropriate bit as determined by the RAC error correction table. As with single-bit error detection, RAC error correction also does not require protection.

Computations on residues are commutative, distributive and associative; therefore RAC can detect and correct for transients inside of the RAC augmented computation block using the reduction logic that is applied to the targeted operation.

B. Components Design, CAD Simulation and Validation for Basic Logic Elements

In this work, simple non-pipelined and non-optimized arithmetic logic adder and multiplier module designs with equally-sized dual 8, 16, 32, 64 and 128-bit operands are implemented with embedded bi-residue RAC capable of detecting multiple bit transients and correcting single bit transients. Figure 3 shows the RAC with EDAC implementation for two-input operations. The A and B inputs as well as the Q output are registered (registers are not depicted in diagram). $M_{0A/B}$ are the residue computation blocks for reducing operands A & B modulo the first modulus M_0 [6]. $M_{0A/B}$ are the residue computation blocks for reducing operands A & B modulo the second modulus M_1. OP is the combinatorial logic function/operation that is RAC augmented. $OP_{M0/1}$ are the operation blocks with operand sizes scaled to that of the moduli. $M_{0/1R}$ are the residue computation blocks for the outputs of $OP_{M0/1}$ blocks. $M_{0/1S}$ are the residue computation blocks for reducing the outputs of the OP blocks modulo the first modulus M_0 and the second modulus M_1. The $\Delta_{M0/1}$ blocks compute the difference in residue magnitude for each modulo between the operation applied to the A & B operands and the operation applied to the residues of the A & B operands. The $M_{0/1D}$ are the residue computation blocks for reducing the outputs of $\Delta_{M0/1}$ blocks modulo the first modulus M_0 and the second modulus M_1.

III. TEST STRUCTURES AND EXPERIMENTAL SETUP

1) Experimental Setup

The arithmetic logic adder and multiplier module designs with bi-residue RAC (single-bit RAC) are synthesized, placed and routed using a generic 45 nm standard cell library that does not include any radiation mitigation or SEE resiliency enhancements. The designs are subjected to rigorous layout-aware fault injection campaigns for ED and EDAC resiliency performance evaluation. Fault injection is performed using a non-commercial SET injection emulation tool. For the purpose of these experiments, only the cell placement and sizing in conjunction with the area overlap of the effective area of a given emulated SET are used to determine the occurrence and duration of an SEU at a given cell node. All cells in the selected technology library are assigned a default SET susceptibility factor that may not reflect the discrete transistor properties and internal layout of the cells. The aggregate affect of node connectedness and placement in the form of individual cell loading and orientation are also not taken into account for these experiments. Throughout all simulations, the arithmetic compute blocks are processing pseudo-random input stimuli that change every clock cycle. Each block processes up to $32x10^6$ bytes of data. The location, start time and duration of all SET strikes are independent events. Up to 500,000 faults are injected into each circuit at a maximum rate of 1 fault every other clock cycle. Each fault can last up to 2 clock cycles. Area overhead, delay, leakage and dynamic power per-computed-word performance metrics are collected using Synopsys and Cadence CAD tools and are compared against that of the equivalent TMR circuits.

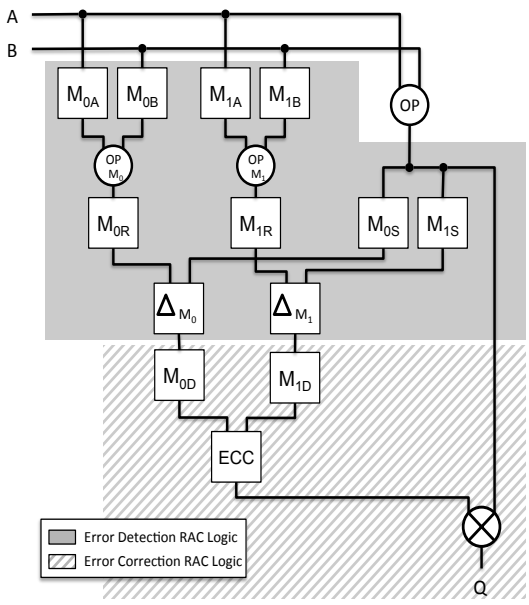

Figure 3 Logic Diagram of ED and EDAC sections of a RAC-enhanced two-input, single-output

IV. RESULTS

Basic arithmetic adder and multiplier blocks were evaluated for SEU mitigation effectiveness in terms of error detection and error correction. Logic transistor count, computation delay and the energy per bit-computed overhead metrics were collected as part of this study. For reference comparison, we also present overhead metrics for TMR (Triple Module Redundancy) hardening approach to SEU mitigation. Figure 4 shows the simulated Error-Detection (ED) and Error-Detection-And-Correction (EDAC) capabilities for a set of Residue-Arithmetic-Code (RAC) hardened standard adders for single-bit errors. The single-bit RAC EDAC enhanced arithmetic blocks detect all single and multi-bit transients and corrects at least 99.19% of all single-bit transients. For multi-bit faults, all accurate single-bit corrections made by the RAC circuitry were counted towards the *Single-bit fault Correction Rate*. The combined correction rate is the aggregate ratio of single-bit transient corrections over the accumulation of single and multi-bit transients. RAC will correct one bit in a multi-bit fault. The selected error-correction scheme determines the correction priority order and can be set independently of the computational significance of output bits in a multi-bit fault. In the advent of a multi-bit fault, it is possible to achieve partial error resiliency and obtain an approximate result by configuring the correction syndrome to address the most significant error-bit. The incidence of multi-bit faults increases with logic depth and connectedness; in the case of arithmetic blocks evaluated, this occurs with increased operand size as shown by the *Combined Correction Rate* for single-bit RAC. The RAC error-correction lookup tables are nodes with $2x[\log_2(M_{0/1})]$ inputs and $[\log_2(M_{0/1})]$ outputs. Upsets to small portions of the aforementioned logic nodes will result in invalid corrections. The proportion of invalid corrections decreases with increased operand sizes as well as with the reduced relative scale of the RAC logic compared to the compute kernel undergoing hardening.

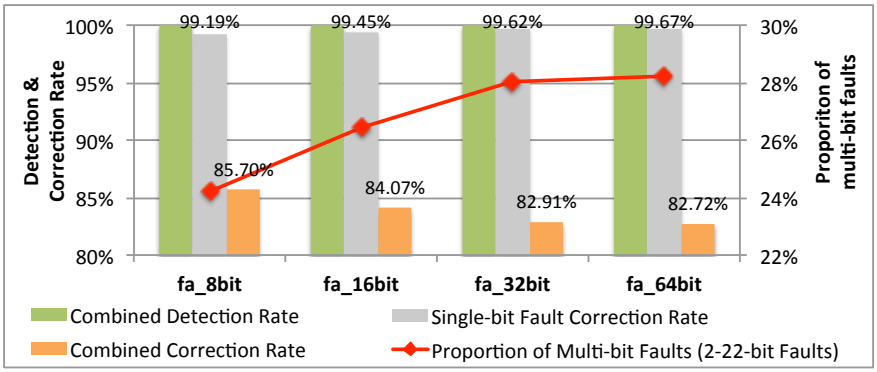

Figure 4 Single-bit RAC ED and EDAC and combined EDAC rates in a 45 nm technology

Figure 5 illustrates the logic overhead requirements for various basic arithmetic blocks when hardened with single-bit RAC EDAC. RAC overhead requirements are driven by the efficiency of the fit of moduli selected relative to the operand sizing as well as by the computational "density" of the operation. As operand sizes are increased, the

difference between the number range representable by the smallest selectable moduli and the compute kernel output range decreases; this results in a more efficient use of the hardening logic. A multiplication operation can be considered "denser" than the equivalent input-operand sized addition, as a multiplication is equivalent to several additions. Consequently, multiple additions are hardened with a single set of moduli in a RAC hardened multiplier module, conversely, a single addition is hardened with the same set of moduli in a RAC hardened adder. The increase in compute kernel density contributes more significantly to the overhead reduction of a RAC hardened solution than the increase in operand size.

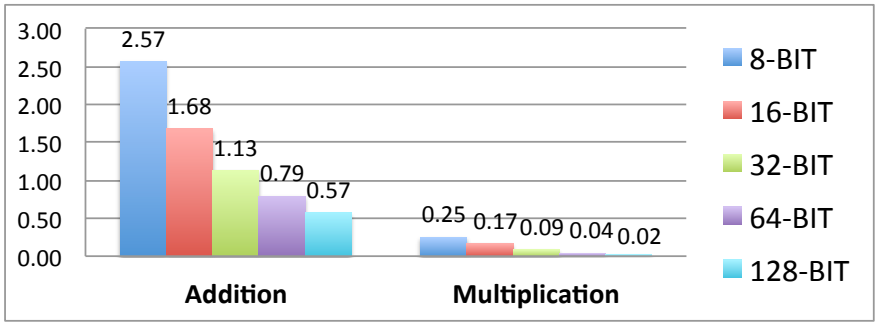

Figure 5 Single-bit RAC-hardened arithmetic block transistor count overhead proportion compared to unhardened equivalent implementation.

Figure 6 compares the computational delay overhead for simple multiplier blocks hardened with RAC against the equivalent blocks hardened with TMR for a 45 nm technology at 0.9V. For RAC hardening, where error detection is performed in parallel to the logic being hardened, any delay in RAC is attributable solely to error correction logic. Error correction logic overhead relative to the combinatorial depth of the logic undergoing hardening decreases with increases in operand sizing for a given compute kernel. RAC delay is 1.05X and 1.42X smaller than that of the equivalent TMR implementation for design in a 45 nm technology operating at 0.9V. Furthermore, unlike temporal filtering, the delay incurred by RAC hardening does not increase with an increase in resiliency.

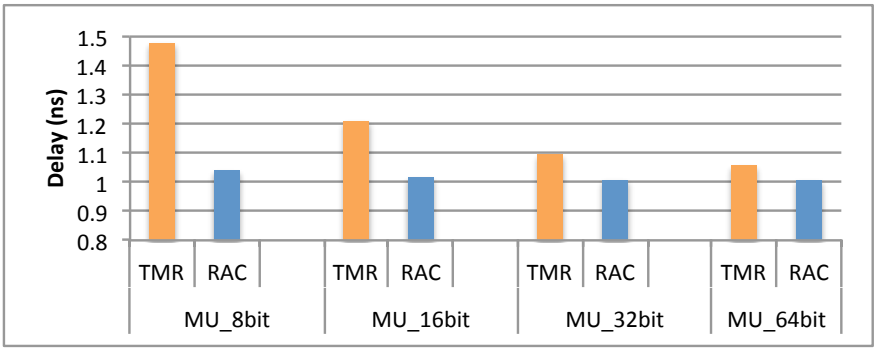

Figure 6 Single-bit RAD EDAC delay overhead of 8, 16, 32 and 64-bit multipliers compared to the equivalent TMR implementation in a 45 nm technology at 0.9 V.

Given the difference in operating speed capabilities between a RAC hardened circuit and the equivalent TMR implementation, the energy per bit computed are compared for

circuit implementations in a 45 nm technology operating at 0.9V. Figure 7 shows that RAC hardening typically requires between 59% and 157% less energy per bit computed than the equivalent TMR implementation.

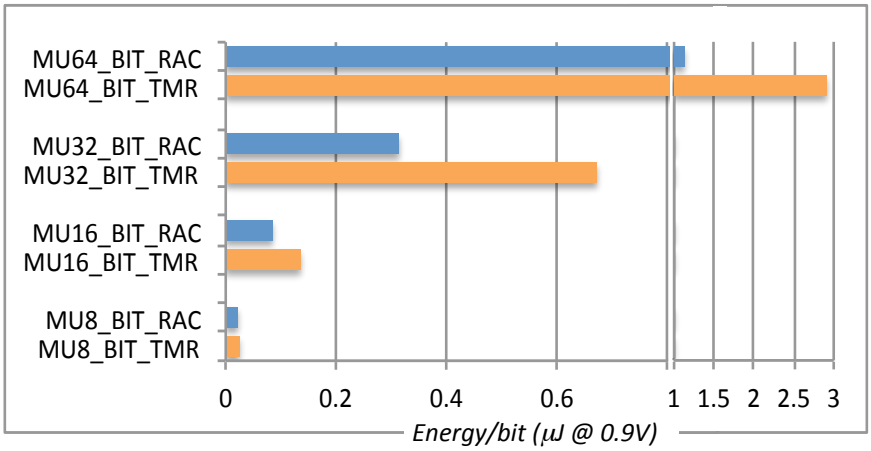

Figure 7 Single-bit RAC EDAC energy per bit computed requirements compared to the equivalent TMR implementation in a 45 nm technology at 0.9 V for 8,16,32, ad 64-bit multiplier unit (MU) sizes

The results show that for appropriately sized moduli, the RAC hardening implemented using a bi-residue code applicable for single bit correction arithmetic logic blocks detects any occurrence of SEUs that result in single or multi-bit faults. In addition to detecting and correcting SEUs that result in faults, RAC logic by way of the computed residue pair difference can also detect any SEU that does not result in a fault. This feature enables RAC hardening logic to act as a low cost Single Event Effect (SEE) strike detector. This SEE detection flag can be used as hardware based "interrupts" to a higher-level control system. The ability to detect non-fault causing SEE events without the requirement of additional logic is a unique feature of RAC.

V. FUTURE WORK

This work developed compelling simulations suggesting that residue codes could be used to detect and correct errors in arithmetic logic circuits. Going forward, the next steps are to complete an iterative experimental-based campaign to first rigorously validate these simulation results on basic arithmetic circuits circuits, and then build this capability into more complex circuits and automated design tools as described below.

Several iterations of design, simulate, build, test, and evaluation are proposed to iteratively build up an understanding of how various arithmetic circuits can be hardened with residue arithmetic coding; starting with simple circuits and then progressing to more complex ones. Some details of these steps are described below. Testing and demonstration using IBM's 32 nm SOI technology are proposed.

For each of the targeted base arithmetic operations, combinations of operand sizes, operand count and desired fault resiliency level, we will develop a mitigation technique by adapting and fitting an appropriate residue code implementation. Each implementation will be expressed using a VHDL netlist. Each netlist implementation will be evaluated for algorithmic correctness, using commercial simulation tools. New netlists representing various architectural organizations will be derived from the original netlists by taking advantage of the commutative, distributive and associative properties of the residue codes and error-correction algorithms. These different architectural implementation netlists will undergo directed graph analysis using the "*radiation characterization tool"* under development for this purpose at ISI. This analysis will yield a functional susceptibility model with a specific architectural vulnerability factor for each netlist by leveraging the error propagation paths and nodal connectedness. The models will be organized as a technology-agnostic "*base element functional model"* library.

In the 32 nm SOI technology, it is necessary to take in to account physical design layout. To this end, various placed and routed instances of the base elements in the "*base element susceptibility model"* library are generated using commercial CAD tools. The resulting layouts are subjected to fault injections using a *"strike emulation tool"* that is developed for this effort. The *"strike emulation tool"* will simulate the affected area(s) of particle strikes in an aggressive radiation environment in order to yield an estimate of design mitigation abilities prior to design fabrication. The *"strike emulation tool"* will apply tessellated shapes representing the area affected by a charge injection into the design layout. The standard cells in the overlapping areas are analyzed using inputs from the "*base element susceptibility model"* and the current state of propagated operand inputs for the simulation. The analysis of each affected cell results in an error either propagating if a propagation threshold is met or an being masked for each standard cell. The overall resiliency performances of the various layout instances are organized in the "*base element physical susceptibility model"* library.

Once designs are fully simulated and then fabricated, the test circuits will undergo single-event testing. The error susceptibility for the design will be characterized in the form of various architectural vulnerability factors (AVF) to be back-annotated into the three levels of models: "*standard cell susceptibility model"* library, "*base element susceptibility model"* library and "*base element physical susceptibility model"* library. The goal is to improve the efficacy of the increasingly higher model levels throughout the successive radiation test campaigns.

The focus of the test campaigns will be to investigate single-event effects using the sources– heavy-ions and protons. Due to the importance of building-upon the simulation models and hardware designs, the campaign will focus on testing and understanding the radiation response of the combinational logic portions of each design. Test campaigns will begin with single-event effect (SEE) static and dynamic characterization at the Naval Research Laboratory using a Two-Photon-Absorption laser system. The laser system is able to emulate a single-event strike by tuning the wavelength of the laser to generate electron-hole pairs as would result from the heavy-ion interactions. Additionally, the strike location of the laser can be controlled and will allow specific targeting of test circuits. From these strikes, USC/ISI will create a database of errors to identify the portions of the RAC circuitry that fail. This information will then be used to calibrate the simulation models and improve the basic element designs.

VI. IMPACT

Developing a process that successfully applies residue arithmetic codes (RAC) to combinatorial elements and mitigates single-event upsets in a radiation environment. This work created a new type of RAC circuit architecture that is highly resilient to single-event upsets (SEU). By building a new architecture, new tools were developed to analyze the effectiveness of the RAC architecture. These tools deconstruct a placed and routed design and perform back-annotated fault injection simulations that give designers the ability to enhance the RAC circuit architecture to increase SEU hardness. Test structures were designed and fabricated in IBM 45 nm SOI to allow a continuation of the work. The single-event test results of these circuits can be compared against other radiation effects work occurring in the 45 nm SOI technology sponsored by DTRA. High impact accomplishments of the CORRECT program are:

- Multiple publications and talks at radiation and government conferences GOMACTech and RADECS.
- Development of residue detection and correction algorithms that can be applied to basic arithmetic components.
- Simulation of residue hardened adders and multipliers in 45 nm SOI that validates the applicability of residue codes as a technique to mitigate single-event upsets in combinatorial logic.

VII. PRIMARY TEAM MEMBERS

USC-ISI, Faculty: Prof. John Granacki
Researchers: Mr. Michel Sika, Dr. Jonathan Ahlbin
Student Interns (Summer 2013): Amir Yazdanbakhsh and Bradley Kiddie

VIII. REFERENCES

[1] Niranjan, S., Frenzel, J.F. "A comparison of fault-tolerant state machine architectures for space-borne electronics", IEEE Transactions on Reliability, Volume 45, Issue 1 p 109-113, 1996

[2] H. Krishna, K. Y. Lin, and J. D. Sun, "A coding theory approach to error control in redundant residue number systems. Part I: Theory and single error correction," IEEE Trans. Circuits Syst., vol. 39, pp. 8-17, Jan. 1992.

[3] Pitsini Mongkolkachit, Bharat Bhuva, "Design Technique for Mitigation of Alpha-Particle-Induced Single-Event Transients in Combinational Logic", Device and Materials Reliability, IEEE Transactions on, Volume: 3, Issue: 3, Sept.2003 Pages: 89- 9

[4] Daniel Lipetz, Eric Schwarz, "Self Checking in Current Floating-Point Units", Computer Arithmetic (ARITH), 2011 20th IEEE Symposium on Digital Object Identifier, 2011

[5] James L. Massey "Survey of residue coding for arithmetic errors" , Int. Computation Cent. Bull. (UNESCO Rome, Italy), vol. 3, no. 4, pp. 3-17, Oct. 1964.

[6] Stanislaw J. Piestrak "Self-Testing Checkers for Arithmetic Codes with Any Check

Base A", Pacific Rim International Symposium on Fault Tolerant Systems, 1991.

IX. DISTRIBUTION LIST

Table 1 CDRL Address List

Office Symbol	Address
DTRA/BE-BLL	8725 John J Kingman Road Fort Belvoir, VA 22060-6201 Willie.Brown@DTRA.mil Dorothy.Russell@dtra.mil
DTRA/BE-BFKR	8725 John J Kingman Road Fort Belvoir, VA 22060-6201 BFKRInbox@dtra.mil
DTRA/BE-BCRT	8725 John J Kingman Road Fort Belvoir, VA 22060-6201 Nazanin CIVILIAN Layne-Cunningham <nazanin.layne-cunningham@dtra.mil>@dtra.mil
DTRA/BE-BI	8725 John J Kingman Road Fort Belvoir, VA 22060-6201 First name.Last name@dtra.mil
DTRA/COR	Defense Threat Reduction Agency/J9NT 8725 John J Kingman Road Fort Belvoir, VA 22060-6201 Leslie.paltuki@dtra.mil

X. PRESENTATIONS

1. M. Sika, J. Ahlbin, M. Bajura, M. Fritze, J. Damoulakis, J. Granacki "Applying Residue Arithmetic Codes to Combinational Logic to Reduce Single Event Upsets," *presented at 2013 Spring Vanderbilt DTRA Review.*

XI. PUBLICATIONS (ATTACHED)

2. M. D Sika, A. Yazdanbakhsh, B. Kiddie, J. R. Ahlbin, M. Fritze, J. Damoulakis, and J. Granacki, "Low Energy Hardening of Combinatorial Logic using Standard Cell s and Residue Codes," *accepted for presentation at 2014 GOMACTech.*
3. M. D Sika, A. Yazdanbakhsh, B. Kiddie, J. R. Ahlbin, M. Fritze, J. Damoulakis, and J. Granacki, "Applying Residue Arithmetic Codes to Combinational Logic to Reduce Single Event Upsets," *2013 IEEE RADECS conference proceedings*.
4. M. Sika, J. Ahlbin, M. Bajura, M. Fritze, J. Damoulakis, "Applying Residue Arithmetic Codes to Combinational Logic to Reduce Single Event Upsets," *2013 GOMACTech Proceedings*.

Applying Residue Arithmetic Codes to Combinational Logic to Reduce Single Event Upsets

Michel D. Sika[1], Jonathan R. Ahlbin[1], Michael Bajura[1], Michael Fritze[1] and John Damoulakis[1]

[1] Information Sciences Institute
University of Southern California
3811 N. Fairfax Dr. Suite 200,
Arlington, VA, USA, 22203-1707

Abstract: Residue arithmetic coding (RAC) is embedded in digital arithmetic logic units to detect and correct single-event upsets (SEUs). Simulations and analyses validate that RAC can be implemented in digital arithmetic logic units.

Keywords: Residue Arithmetic coding; Residue Checking System; Radiation Hardening; Self Checking; Self Correcting; Error Detection; Rad-hard-by-design.

Introduction

Block-code (i.e. EDAC) based error detection and correction techniques, and circuit-centric radiation hardening methods have been effective to mitigate single-event upsets (SEUs) in memory elements. An effective efficient implementation of SEU mitigation techniques in the processing components of a computing chain, however, has remained elusive. This work shows that implementing residue-arithmetic coding (RAC) may be an effective means to harden the arithmetic logic processing components of a system from SEUs without requiring triple modular redundancy or similar approaches with their high overhead in terms of area, speed, power and upset susceptibility.

Background

Typically, system level SEU fault tolerance is achieved for logic operations with duplication techniques such as Triple Modular redundancy (TMR) [1]. Unfortunately, a TMR system implementation results in at least a 3x increase in area and power overhead for the protected area. Compared to TMR, RAC is a promising alternative lower overhead computation method that may enable the concurrent detection of errors in logic (arithmetic) computations with lower overhead penalties.

A residue is the remainder from the operation of dividing an integer by a modulus. For example, the residue of an n-bit operand A modulo m ($[A]_m$) is the remainder obtained from dividing A by m. The choice of m (or m's), and the resulting residue value $[A]_m$ (or values $[A]_{m1}$, $[A]_{m2}$), can be used to uniquely characterize error patterns in an arithmetic operation [2][3].

RAC Implementation

Residue arithmetic coding for error detection are computations on the residues of operands for a given modulus or set of moduli, the outcome of which are compared with the residues of the standard operation reduced modulo the same set of moduli. The number of moduli employed as well as the value of these moduli determines the "strength" of the RAC, i.e. 1 or 2-bit detection and/or correction. Figure 1 schematically presents the application of residue coding to the hardening of an N-bit ADDER for 1-bit detection and correction capability [4].

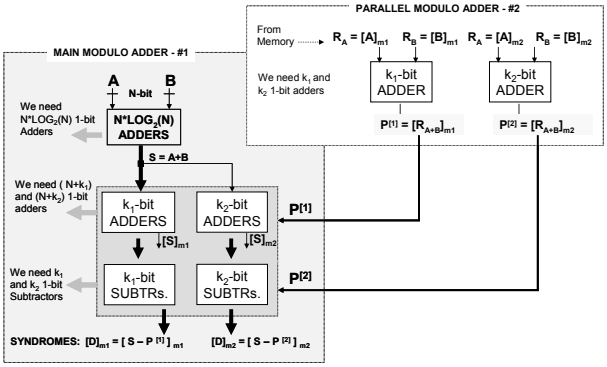

Figure 1. 1-bit ED and EDAC applied to an N-bit adder using 2 moduli.

Example Modulus Calculation: For a given modulus m and a simple 2-input 1-output operation with operands A, B and output S, we compute

$[A]_m$, $[B]_m$ and $[Q]_m$

We have for an arbitrary operation *OP*:

(1) $[A]_m \; OP \; [B]_m = [Q]_m$

If there is a 1-bit error then:

(2) $[A]_m \; OP \; [B]_m = [Q]_m \pm OP \; [2^i]_m$

Where *i* indicates the position of the error reflected in the *Q* side of the operation. We can see from this example that it is possible to detect a single error with a single modulus. However, it is not possible to reliably find and correct the bit with the error as 2 or more *i* can be congruent modulo *m*. If we introduce a second modulus *n* however, and designate *r* and *s* as 2 *i* such that:

(3) $r \equiv s \bmod(m)$

then (3) implies that we can not have:

(4) $r \equiv s \bmod(n)$

for *n* and *m* that are relatively prime and each with magnitude in excess of the maximum range for *r* and *s* [5].

With the above conditions we can therefore find the position of a single bit error with two moduli.

Computations on residues are commutative, distributive and associative; therefore RAC can detect and correct for transients inside of the RAC augmented computation block using the reduction logic that is applied to the targeted operation.

RAC Design

In this work, two types of RAC designs are evaluated for both error detection (ED) and EDAC against a non-RAC augmented design. The designs are fundamental arithmetic modules consisting of an adder or multiplier with predetermined operand sizes. The initial adder or multiplier arithmetic block design is augmented with the full RAC-based EDAC for SEU faults. Figure 2 shows the components required for a non-optimized RAC augmentation of a Given Operation.

Figure 2 is an illustration showing how RAC with EDAC is implemented for 2 input operations. The *A* and *B* inputs as well as the *Q* output are registered (registers are not depicted in diagram). $M_{0A/B}$ are the residue computation blocks for reducing operands A & B modulo the first modulus M_0 [6]. $M_{0A/B}$ are the residue computation blocks for reducing operands A & B modulo the second modulus M_1. *OP* is the Combinatorial Logic Function/Operation that is RAC augmented. $OP_{M0/1}$ are the Operation blocks with operand sizes scaled to that of the moduli. $M_{0/1R}$ are the residue computation blocks for the outputs of $OP_{M0/1}$ blocks. $M_{0/1S}$ are the residue computation blocks for reducing the outputs of the *OP* blocks modulo the first modulus M_0 and the second modulus M_1. The $\Delta_{M0/1}$ blocks compute the difference in residue magnitude for each modulo between the Operation applied to the A & B operands and the Operation applied to the residues of the A & B operands. The $M_{0/1D}$ are the residue computation blocks for reducing the outputs of $\Delta_{M0/1}$ blocks modulo the first modulus M_0 and the second modulus M_1.

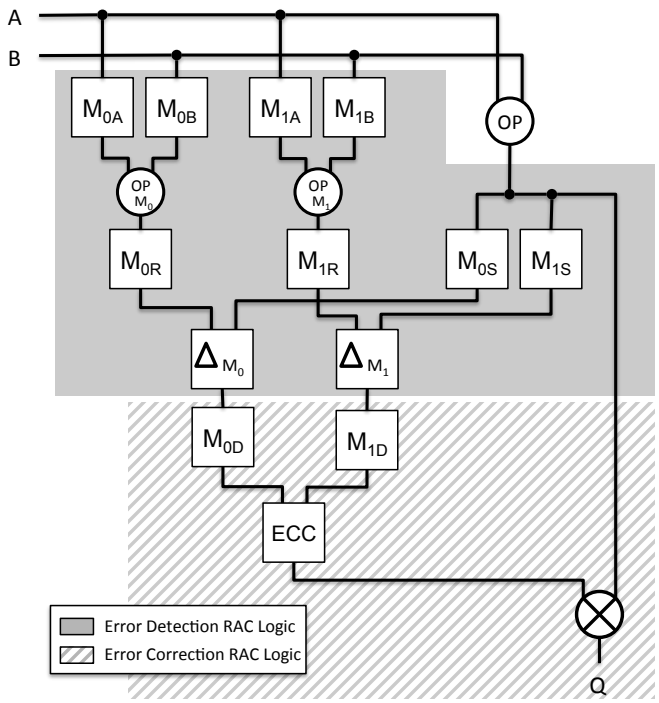

Figure 2. Logic Diagram of ED and EDAC sections of a RAC enhanced 2-input, 1-output operation.

Simulating the RAC Augmented Designs

Simulation Setup: A simple experiment is established to verify the functionality of the RAC design and to investigate the following: 1) The ability of a RAC enhanced circuit to detect single-bit errors, 2) The ability of a RAC enhanced circuit to correct single-bit errors. To stimulate these designs, fault injection hooks are added to all possible insertion points of the arithmetic blocks. A basic fault injector program with an exhaustive set of test patterns for single SEUs is developed and applied to the arithmetic blocks through the basic fault injector program. When subjected to single upsets, these systematic simulations give insight into the effectiveness, overhead, and performance for basic RAC enhanced elementary modules.

This initial study does not consider any physical placement, feature size or arithmetic module functional use patterns. The study is also independent of any specific fabrication technology or implementation platform. The Operation module that is to be augmented is described in HDL using generic operand sizes. All the RAC logic modules as well as the augmentation process are also described in generated HDL. The RAC augmented logic

test pattern generation, insertion and simulation for SEUs is produced and operated by the RAC augmentation process.

A node is defined as any input or output to a logic gate. For instance, a 3 input NAND gate has 4 nodes, 3 input nodes and 1 output node. For the purpose of this simple experiment, a fault injection epoch is defined as an arbitrary number between 2 and 10 system clock periods. All fault injection epochs are synchronous with the system clock. For each epoch, a single node is selected at random. The adder processes approximately 47.5 million bytes of arbitrary data during the experiment. Results are shown for two instances of a non-optimized RAC enhanced simple 8-bit adder were subjected to 1,000,000 single node flips.

Given that the injected bit flips are pseudo-random and are not correlated to targeted stages of an Operation nor to specific operand sets, not all single node-flips result in faults that would be latched in the Operation output registers set. Node-flips that do result in faults are designated as *Faults Injected.*

A *Fault Detected* is defined as an instance where the result produced by the plain Operation logic does not match the expected golden result AND the RAC logic generates a non-zero flag value. Non-zero RAC flag values indicate that the RAC logic estimates that an error has occurred either in the Operation Logic or within the RAC logic. Faults that affect the logic that generates the RAC flags will go undetected and uncorrected by RAC.

Faults requiring correction (FRC) are defined as faults that occurred in the Operation logic that require error correction, FRCs occur when the golden value and the non-RAC corrected value of an Operation for a given set of operands differ. The RAC logic does not successfully detect all the FRCs; the portion of faults that are caught and accurately corrected for are labeled FCA for *Faults with Correction Applied*. Faults that are either not caught by the RAC logic, or are valid FCAs to which an erroneous correction is applied or altogether invalid FCAs (false positive) are considered *Missed*.

Fault Injection Results: Figure 3 depicts the first instance of the RAC enhanced adder. The RAC logic set in combinatorial logic results in a ~2.7x logic overhead compared to a non-RAC enhanced adder. Figure 4 depicts the second instance of the RAC enhanced adder; this design implements the RAC residue computation logic and error correction table in hardened lookup tables. By using hardened error correction lookup tables, there is a significant reduction in the logic overhead to ~1.5x compared to the previous design. The two designs are otherwise identical in terms of RAC control and differentiation logic.

The specific type of radiation hardened state element used for the residue lookup tables in Figure 4 is not relevant. The experiment is designed in a manner such that no logic state within the lookup tables is affected by the fault injection. For this experiment, the residue and ECC lookup tables are essentially nodes with $2x[\log_2(M_{0/1})]$ inputs and $[\log_2(M_{0/1})]$ outputs. These node inputs and outputs are subject to bit flips just like the ports on any other node.

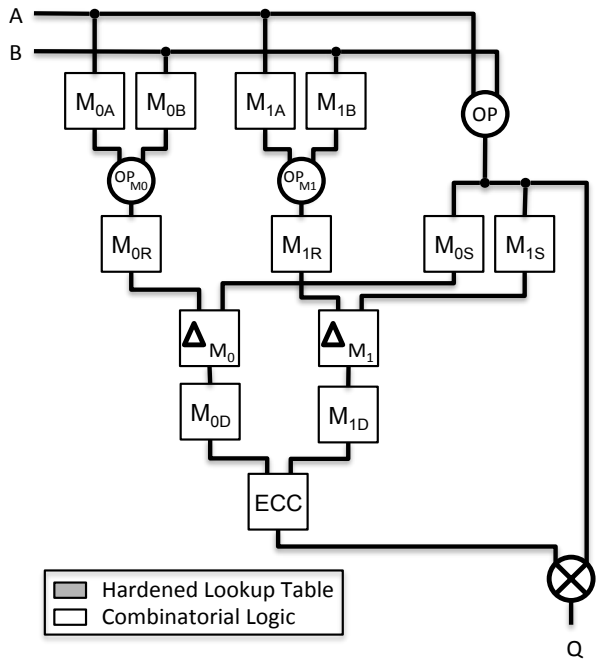

Figure 3. Logic Diagram of a RAC enhanced 2-input, 1-output operation with RAC circuitry entirely implemented in combinatorial logic.

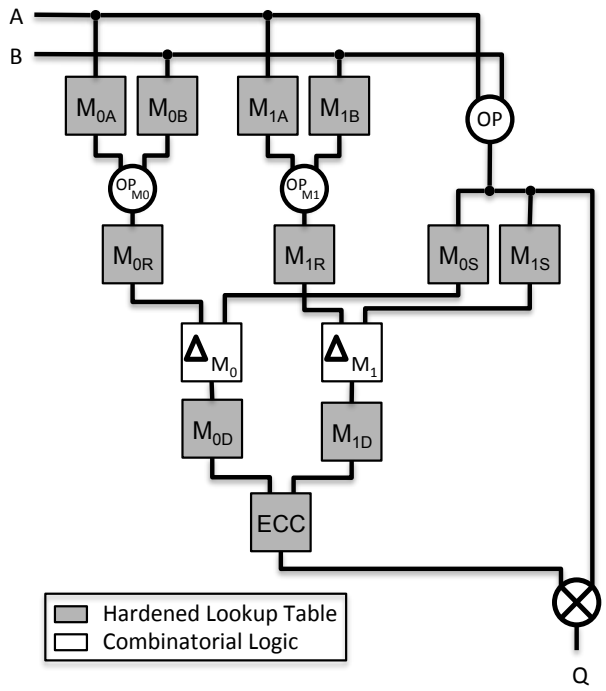

Figure 4. Logic Diagram of a RAC enhanced 2-input, 1-output operation with RAC residue circuitry implemented in combinatorial Radiation Hardened Lookup Tables

The resiliency results for the implementation in Figure 3 are summarized in Table 1.

Full RAC in Combinatorial Logic		
Metric	**Count**	**Rate**
Faults Injected	90,242	NA
Faults Detected	90,026	99.76%
FRC	52,884	58.74%
FCA	43,305	81.87%
Missed	216	0.24%

Table 1. Resiliency of full-combinatorial RAC augmented 8-bit general-purpose full adder when subjected to 1,000,000 single node flips and processing 47.5 million bytes of data. *Faults Injected* represents the number of single bit errors that would be latched in output registers without RAC. *Faults Detected* is the number of errors correctly detected by the RAC logic. *FRC* is the number of faults correctly determined by RAC as needing error corrections. *FCA* is the number of faults requiring correction that RAC successfully fixed. *Missed* is the number of faults that are neither detected nor corrected by RAC.

The resiliency results for the implementation in Figure 4 are summarized in Table 2.

Partial RAC in Combinatorial Logic		
Metric	**Count**	**Rate**
Faults Injected	66,066	NA
Faults Detected	65,984	99.88%
FRC	39,304	59.57%
FCA	29,873	76.00%
Missed	82	0.12%

Table 2. Resiliency of partial-combinatorial RAC augmented 8-bit general-purpose full adder when subjected 1,000,000 single node flips and processing 47.5 million bytes of data. *Faults Injected* represents the number of single bit errors that would be latched in output registers without RAC. *Faults Detected* is the number of errors correctly detected by the RAC logic. *FRC* is the number of faults correctly determined by RAC as needing error corrections. *FCA* is the number of faults requiring correction that RAC successfully fixed. *Missed* is the number of faults that are neither detected nor corrected by RAC.

The results show that for appropriately sized moduli, the RAC logic detects more than 99% injected faults that would result in an incorrect value being latched at the Q output.

Given that a relatively small amount of logic is required for the fault indication and flag generation circuitry, faults in this portion of the RAC logic are a rare occurrence (< 0.25%); RAC can not however correct for these faults.. RAC only accurately determines that 59% to 60% of FRC as actually needing correction. Of theses faults determined as needing correction, RAC accurately corrects 76% to 82% of errors. Given that RAC is essentially a reduction logic function, where a relatively large set of combined inputs are reduced to a small output, RAC ED is very sensitive to any faults that target any RAC enhanced computation modules. A side effect of this sensitivity is the limited ability of this non-optimized RAC implementation to perform accurate error correction. The partial-combinatorial RAC augmented circuit greatly reduces the logic overhead requirements compared to the non-optimized fully combinatorial implementation. The fault detection rate for the reduced overhead circuit is only slightly improved whereas the accuracy of fault corrections is deteriorated by almost 6 percentage points.

Conclusions

By performing aggressive SEU fault injections simulations for single bit-transients we have shown that RAC looks to be an attractive, lower overhead, alternative to triple modular redundancy for SEU hardening of logic operations. RAC is capable of detecting over 99% of

faults with as little as 50% overhead for a non-optimized implementation.

Acknowledgements

The authors would like to acknowledge the DTRA–CORRECT program for their financial support. (Contract #: HDTRA1-12-C-0057).

References:

[1] Niranjan, S., Frenzel, J.F. "A comparison of fault-tolerant state machine architectures for space-borne electronics", IEEE Transactions on Reliability, Volume 45, Issue 1 p 109-113, 1996

[2] H. Krishna, K. Y. Lin, and J. D. Sun, "A coding theory approach to error control in redundant residue number systems. Part I: Theory and single error correction," IEEE Trans. Circuits Syst., vol. 39, pp. 8-17, Jan. 1992.

[3] Daniel Lipetz, Eric Schwarz, "Self Checking in Current Floating-Point Units", Computer Arithmetic (ARITH), 2011 20th IEEE Symposium on Digital Object Identifier, 2011

[4] James L. Massey "Survey of residue coding for arithmetic errors" , Int. Computation Cent. Bull. (UNESCO Rome, Italy), vol. 3, no. 4, pp. 3-17, Oct. 1964.

[5] Stanislaw J. Piestrak "Self-Testing Checkers for Arithmetic Codes with Any Check Base A", Fault Tolerant Systems, 1991. Proceedings., Pacific Rim International Symposium on

Applying Residue Arithmetic Codes to Combinational Logic to Reduce Single Event Upsets

M. D. Sika, A. Yazdanbakhsh, B. Kiddie,
J. R. Ahlbin, M. Bajura, M. Fritze, J. Damoulakis and J. Granacki

***Abstract* — Mitigating Single Event Upsets (SEU) in combinatorial logic is conventionally accomplished through redundancy based Radiation Hardening By Design (RHBD) methods such as Triple Module Redundancy (TMR). A hardening technique based on residue arithmetic codes (RAC) is proposed as a lower overhead alternative for detecting and correcting SEUs in arithmetic logic units. Simulations and analyses at the 45nm node show that RAC detects over 99% of faults with 2.6X less area and 157% less energy than TMR.**

***Index Terms*— Residue codes; Arithmetic error checking system; Radiation Hardening; Self Checking; Self Correcting; Error Detection; RHBD; SEU.**

I. Introduction

In combinatorial logic, the error cross section of a circuit increases proportionately to the increase in Single Event Transient (SET) width [1]. As technology nodes reduce in scale, the effects of SETs will result in an increased occurrence of SEUs. While Block-code based error detection and correction techniques and circuit-centric radiation hardening methods are effective to mitigate SEUs in memory elements, a reliable and low overhead SEU mitigation technique for combinatorial logic has remained elusive. This work shows that residue codes applied to arithmetic blocks or residue-arithmetic coding (RAC) are an effective means to harden the arithmetic logic processing components of a system from SEUs without the high area, speed, power requirements of triple modular redundancy. Residue codes are introduced followed by a description of the RAC architecture. The fault injection method and experimental setup are subsequently described. SEU error detection and error correction effectiveness is assessed using fault injection simulations of fully placed and routed simple arithmetic blocks designed for single-bit correction in a 45nm technology. Logic overhead, delay propagation and power metrics are also collected.

This work was supported (in part) by the Defense Threat Reduction Agency, Basic Research Award # HDTRA1-12-C-0057, to the USC Information Sciences Institute and the University of Southern California.

M. D. Sika, J. R. Ahlbin, M. Bajura, M. Fritze, J. Damoulakis, and J. Granacki are with University of Southern California, Information Sciences Institute, 3811 N. Fairfax Drive, Suite 200, Arlington VA, 22203-1707, (703)-248-6168, msika@isi.edu.

A. Yazdanbakhsh is with the University of Wisconsin, 1415 Engineering Drive, Madison, WI 53706

B. Kiddie is with Vanderbilt University, Box 1824 Station B. Nashville, TN 37235

II. Background

In combinatorial logic, system level SEU fault tolerance is typically achieved with either spatial techniques based on duplication such as Triple Modular redundancy (TMR) [2] or delay based transient filtering such as temporal filtering [3]. Unfortunately the implementation of TMR requires an area overhead penalty of at least 3x in addition to a significant power overhead requirement for the protected area. Temporal filtering adds to the logic overhead in addition to limiting circuit operating frequencies. Either method or combination of methods diminish the power and area benefits of designing circuits in small technology nodes at lower voltages. Residue code based error correction schemes were shown to be an effective low cost scheme to improve resiliency in the combinatorial logic of communication systems [5].

The Residue Arithmetic Code or RAC is based on the concurrent detection of faults using operations on residue-coded operands. Correction is achieved with minor additional delay by evaluating the outcome of the operations on the residue-coded operands against the residue-coded result of the computation component undergoing hardening. The ability of residue codes to perform concurrent operations on residue-coded operands is what presents RAC with the ability to detect and correct with much lower overhead than redundancy based RHBD techniques and less delay than temporal filtering techniques. RAC for error detection (ED) or error detection and correction (EDAC) are computations on the residues of the set of operands R for a given modulus or set of moduli M.

A residue is the remainder from the operation of dividing an integer by a modulus. The residue of an n-bit operand A modulo m labeled $[A]_m$, is the remainder obtained from dividing A by m. If R is the set of operands where $R=\{A, B, \ldots\}$ and M the set of moduli where $M=\{m_0, m_1, \ldots\}$ then R_M is the set of resulting residues $R_M = \{\{[A]_{m0}, [A]_{m1} \ldots\}, \{[B]_{m0}, [b]_{m1} \ldots\}, \ldots\}$ where each operand is reduced modulo each modulus. If the compute kernel has inputs in the form of operands R and yields an output labeled S, then S_M is the compute kernel output reduced modulo M. The sets of M and

R_M are used to uniquely characterize error patterns in an arithmetic operation [3][4]. R_M is compared with the residue set S_M in order to detect and correct SEU errors. The number and arithmetic properties of the selected M determines the "arithmetic weight" of the errors that RAC can detect and/or correct, and consequently the EDAC "strength" of the code. Although a single modulus is appropriate for error detection, error correction with a single modulus is not reliable unless the modulus does not result in congruencies when comparing R_M and S_M. Such a modulus however would need to be significantly larger in terms of number of bits compared to the size of the output of the compute kernel that is to be hardened. For reduced overhead and improved efficiency in error correction, an additional modulus is used for every bit of correction that is required in lieu of a single large modulus. Single-bit error correction requires the selection of 2 moduli, dual-bit error correction requires the selection of 3 moduli, et cetera. This work evaluates the effectiveness of two-moduli based RAC designed for single-bit correction and multi-bit detection, this is known as a biresidue code.

Figure 1 schematically presents the application of residue coding to the hardening of an N-bit ADDER for 1-bit detection and correction capability [4].

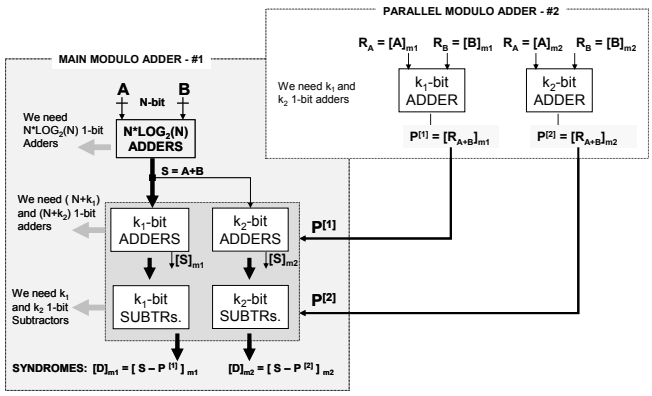

Fig. 1. 1-bit ED and EDAC applied to an N-bit adder using 2 moduli.

Example RAC computation:

For a given modulus m and a simple 2-input 1-output operation *OP* with operands A, B and output S, we have

We compute:

$[A]_m$, $[B]_m$ and $[S]_m$

We have for an arbitrary operation *OP*:

(1) $A \; OP \; B = S$
(2) $[A]_m \; OP \; [B]_m = [S]_m$

If there is a 1-bit error then:

(3) $[A]_m \; OP \; [B]_m = [Q]_m \pm OP \; [2^i]_m$

Where i indicates the position of the error reflected in the S side of the operation. The example shows that it is possible to detect a single error with a single modulus. However, it is not possible to reliably find and correct the bit with the error, as 2 or more i can be congruent modulo m. If we introduce a second modulus n however, and designate r and s as $2i$ such that:

(4) $r \equiv s \bmod(m)$

then (4) implies that we can not have:

(5) $r \equiv s \bmod(n)$

for n and m that are relatively prime and each with magnitude in excess of the maximum range for r and s [5].

With the above conditions we can find the position of a single bit error with two moduli. The residue sets of the operands and the computation result are compared pairwise for each modulus. If both pairs of residue differences do not match, error correction is implemented by flipping the appropriate bit as determined by the RAC error correction table.

Computations on residues are commutative, distributive and associative; therefore RAC can detect and correct for transients inside of the RAC augmented computation block using the reduction logic that is applied to the targeted operation.

III. Integrating RAC into a Digital Design

In this work, simple non-pipelined and non-optimized arithmetic logic adder and multiplier module designs with equally sized dual 8, 16, 32, 64 and 128-bit operands are implemented with embedded biresidue RAC capable of detecting multiple bit transients and correcting single bit transients. Figure 2 shows the RAC with EDAC implementation for 2 input operations. The A and B inputs as well as the Q output are registered (registers are not depicted in diagram). $M_{0A/B}$ are the residue computation blocks for reducing operands A & B modulo the first modulus M_0 [6]. $M_{0A/B}$ are the residue computation blocks for reducing operands A & B modulo the second modulus M_1. *OP* is the combinatorial logic function/operation that is RAC augmented. $OP_{M0/1}$ are the operation blocks with operand sizes scaled to that of the moduli. $M_{0/1R}$ are the residue computation blocks for the outputs of $OP_{M0/1}$ blocks. $M_{0/1S}$ are the residue computation blocks for reducing the outputs of the *OP* blocks modulo the first modulus M_0 and the second modulus M_1. The $\Delta_{M0/1}$ blocks compute the difference in residue magnitude for each modulo between the operation applied to the A & B operands and the operation applied to the residues of the A & B operands. The $M_{0/1D}$ are the residue computation blocks for reducing the outputs of $\Delta_{M0/1}$ blocks modulo the first modulus M_0 and the second modulus M_1.

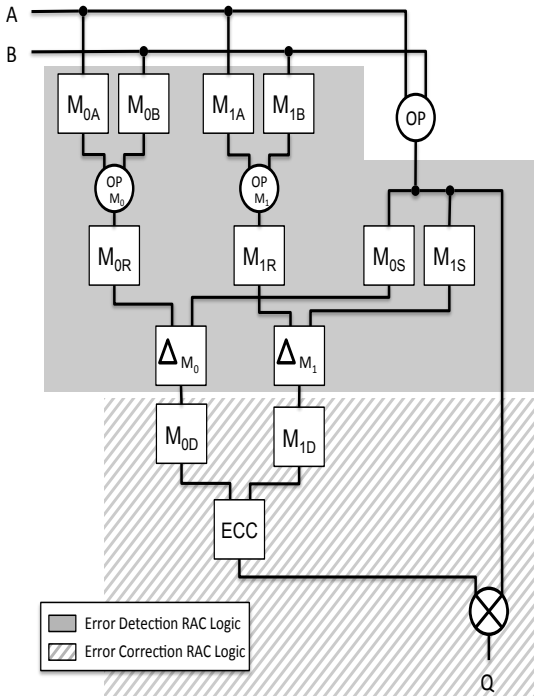

Fig. 2. Logic Diagram of ED and EDAC sections of a RAC enhanced 2-input, 1-output operation.

IV. Experimental Setup

The arithmetic logic adder and multiplier module designs with biresidue RAC (1-bit RAC) are synthesized, placed and routed using a generic 45nm standard cell library that does not include any radiation mitigation or SEE resiliency enhancements. The designs are subjected to rigorous layout aware fault injection campaigns for ED and EDAC resiliency performance evaluation. Fault injection is performed using a non-commercial SET injection emulation tool. For the purpose of these experiments, only the cell placement and sizing in conjunction with the area overlap of the effective area of a given emulated SET are used to determine the occurrence and duration of an SEU at a given cell node. All cells in the selected technology library are assigned a default SET susceptibility factor that may not reflect the discrete transistor properties and internal layout of the cells. The aggregate affect of node connectedness and placement in the form of individual cell loading and orientation are also not taken into account for these experiments. Throughout all simulations, the arithmetic compute blocks are processing pseudo-random input stimuli that change every clock cycle. Each block processes up to 32.10^6 bytes of data. The location, start time and duration of all SET strikes are independent events. Up to 500,000 faults are injected into each circuit at a maximum rate of 1 fault every other clock cycle. Each fault can last up to 2 clock cycles. Area overhead, delay, leakage and dynamic power per-computed-word performance metrics are collected using Synopsys and Cadence CAD tools and are compared against that of the equivalent TMR circuits.

V. Results and Performance

A. RAC Resiliency Performance

Figure 3 shows the ED and EDAC capabilities for a set of RAC hardened standard adders. The single-bit EDAC RAC logic designs detect 100% of all single and multi-bit transients and correct at least 99.19% of all single-bit transients. The combined correction rate is the aggregate ratio of 1-bit transient corrections over the accumulation of single and multi-bit transients. RAC will correct 1-bit in a multi-bit fault. The selected error correction scheme determines the correction priority order and can be set independently of the computational significance of output bits in a multi-bit fault. The incidence of multi-bit faults increases with logic depth and connectedness; in the case of arithmetic blocks evaluated, this occurs with increased operand size. The RAC error correction lookup tables are nodes with $2x[\log_2(M_{0/1})]$ inputs and $[\log_2(M_{0/1})]$ outputs. Upsets to small portions of the aforementioned logic nodes will result in invalid corrections. The proportion of invalid corrections decreases with increased operand sizes as well as with the reduced relative scale of the RAC logic compared to the compute kernel undergoing hardening.

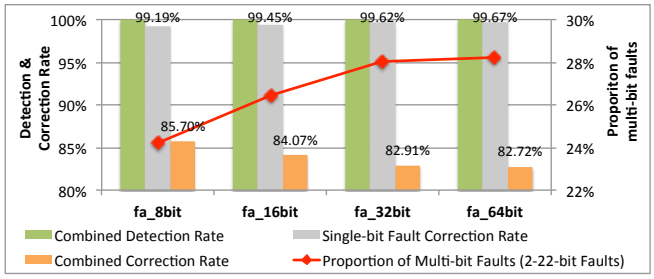

Fig. 3. 1-bit RAC ED and EDAC and combined EDAC rates in a 45nm technology.

B. RAC Logic Overhead

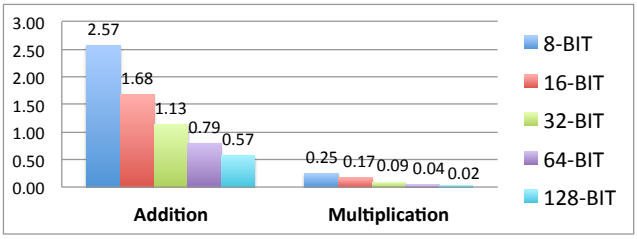

Fig. 4. 1-bit RAC hardened arithmetic block logic overhead requirements

Figure 4 illustrates the logic overhead requirements for various basic arithmetic blocks when hardened with single-bit RAC EDAC. RAC overhead requirements are driven by the efficiency of the fit of moduli selected relative to the operand sizing as well as by the computational "density" of the operation. As operand sizes are increased, the difference between the number range representable by the smallest selectable moduli and the compute kernel output range decreases; this results in a more efficient use of the hardening logic. A multiplication operation can be considered "denser" than the equivalent input-operand sized addition, as a

multiplication is equivalent to several additions. Consequently, multiple additions are hardened with a single set of moduli in a RAC hardened multiplier module, conversely, a single addition is hardened with the same set of moduli in a RAC hardened adder. The increase in compute kernel density contributes more significantly to the overhead reduction of a RAC hardened solution than the increase in operand size.

C. RAC Delay overhead

Figure 5 compares the computational delay overhead for simple multiplier blocks hardened with RAC against the equivalent blocks hardened with TMR for a 45nm technology at 0.9V. For RAC hardening, where error detection is performed in parallel to the logic being hardened, any delay in RAC is attributable solely to error correction logic. Error correction logic overhead relative to the combinatorial depth of the logic undergoing hardening decreases with increases in operand sizing for a given compute kernel. RAC delay is 1.05X and 1.42X smaller that that of the equivalent TMR implementation for design in a 45nm technology operating at 0.9V. Furthermore, unlike temporal filtering, the delay incurred by RAC hardening does not increase with an increase in resiliency.

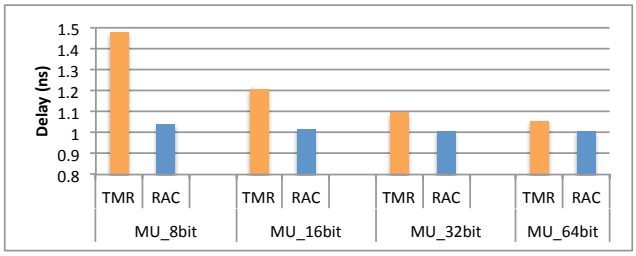

Fig. 5. Single-bit RAC EDAC delay overhead compared to the equivalent TMR implementation in a 45nm technology at 0.9V.

D. RAC Energy overhead

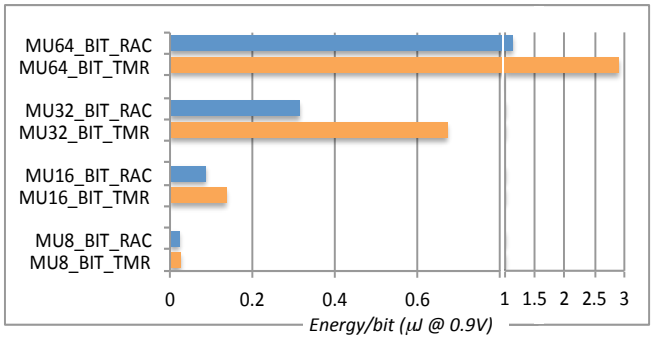

Fig. 6. Single-bit RAC EDAC energy per bit computed requirements compared to the equivalent TMR implementation in a 45nm technology at 0.9V.

Given the difference in operating speed capabilities between a RAC hardened circuit and the equivalent TMR implementation, the energy per bit computed are compared for circuit implementations in a 45nm technology operating at 0.9V instead of power. Figure 6 shows that RAC hardening typically requires between 59% and 157% less energy per bit computed than the equivalent TMR implementation.

VI. Discussion

The results show that for appropriately sized moduli, the RAC hardening implemented using a biresidue code applicable for single bit correction arithmetic logic blocks detects any occurrence of SEUs that result in faults single or multi-bit faults. In addition to detecting and correcting SEUs that result in faults, RAC logic by way of the computed residue pair difference can also detect any SEU that does not result in a fault. This feature enables RAC hardening logic to act as a low cost Single Event Effect (SEE) strike detector. This SEE detection flag can be used as hardware based "interrupts" to a higher-level control system. The ability to detect non-fault causing SEE events without the requirement of additional logic is a unique feature of RAC.

VII. Conclusion

SET injection emulation based simulations show that RAC is an attractive, low overhead, alternative to triple modular redundancy for SEU hardening of combinatorial logic based arithmetic modules in a 45nm technology. RAC is capable of detecting over 99% of faults. Single-bit RAC EDAC hardening requires 2.6X less area operates 42% faster and requires 157% less Energy than TMR. Furthermore, RAC circuitry can serve as SEE environment sensors.

Acknowledgements

This work was supported (in part) by the Defense Threat Reduction Agency, Basic Research Award # HDTRA1-12-C-0057, to the USC Information Sciences Institute and the University of Southern California.

References

[1] Niranjan, S., Frenzel, J.F. "A comparison of fault-tolerant state machine architectures for space-borne electronics", IEEE Transactions on Reliability, Volume 45, Issue 1 p 109-113, 1996

[2] H. Krishna, K. Y. Lin, and J. D. Sun, "A coding theory approach to error control in redundant residue number systems. Part I: Theory and single error correction," IEEE Trans. Circuits Syst., vol. 39, pp. 8-17, Jan. 1992.

[3] Pitsini Mongkolkachit, Bharat Bhuva, "Design Technique for Mitigation of Alpha-Particle-Induced Single-Event Transients in Combinational Logic", Device and Materials Reliability, IEEE Transactions on, Volume: 3, Issue: 3, Sept.2003 Pages: 89- 9

[4] Daniel Lipetz, Eric Schwarz, "Self Checking in Current Floating-Point Units", Computer Arithmetic (ARITH), 2011 20th IEEE Symposium on Digital Object Identifier, 2011

[5] James L. Massey "Survey of residue coding for arithmetic errors" , Int. Computation Cent. Bull. (UNESCO Rome, Italy), vol. 3, no. 4, pp. 3-17, Oct. 1964.

[6] Stanislaw J. Piestrak "Self-Testing Checkers for Arithmetic Codes with Any Check Base A", Pacific Rim International Symposium on Fault Tolerant Systems, 1991.

Applying Residue Arithmetic Codes to Combinational Logic to Reduce Single Event Upsets

Michel D. Sika[1], Jonathan R. Ahlbin[1], Michael Bajura[1], Michael Fritze[1] and John Damoulakis[1]

[1] Information Sciences Institute
University of Southern California
3811 N. Fairfax Dr. Suite 200,
Arlington, VA, USA, 22203-1707

Abstract: Residue arithmetic coding (RAC) is embedded in digital arithmetic logic units to detect and correct single-event upsets (SEUs). Simulations and analyses validate that RAC can be implemented in digital arithmetic logic units.

Keywords: Residue Arithmetic coding; Residue Checking System; Radiation Hardening; Self Checking; Self Correcting; Error Detection; Rad-hard-by-design.

Introduction

Block-code (i.e. EDAC) based error detection and correction techniques, and circuit-centric radiation hardening methods have been effective to mitigate single-event upsets (SEUs) in memory elements. An effective efficient implementation of SEU mitigation techniques in the processing components of a computing chain, however, has remained elusive. This work shows that implementing residue-arithmetic coding (RAC) may be an effective means to harden the arithmetic logic processing components of a system from SEUs without requiring triple modular redundancy or similar approaches with their high overhead in terms of area, speed, power and upset susceptibility.

Background

Typically, system level SEU fault tolerance is achieved for logic operations with duplication techniques such as Triple Modular redundancy (TMR) [1]. Unfortunately, a TMR system implementation results in at least a 3x increase in area and power overhead for the protected area. Compared to TMR, RAC is a promising alternative lower overhead computation method that may enable the concurrent detection of errors in logic (arithmetic) computations with lower overhead penalties.

A residue is the remainder from the operation of dividing an integer by a modulus. For example, the residue of an n-bit operand A modulo m ($[A]_m$) is the remainder obtained from dividing A by m. The choice of m (or m's), and the resulting residue value $[A]_m$ (or values $[A]_{m1}$, $[A]_{m2}$), can be used to uniquely characterize error patterns in an arithmetic operation [2][3].

RAC Implementation

Residue arithmetic coding for error detection are computations on the residues of operands for a given modulus or set of moduli, the outcome of which are compared with the residues of the standard operation reduced modulo the same set of moduli. The number of moduli employed as well as the value of these moduli determines the "strength" of the RAC, i.e. 1 or 2-bit detection and/or correction. Figure 1 schematically presents the application of residue coding to the hardening of an N-bit ADDER for 1-bit detection and correction capability [4].

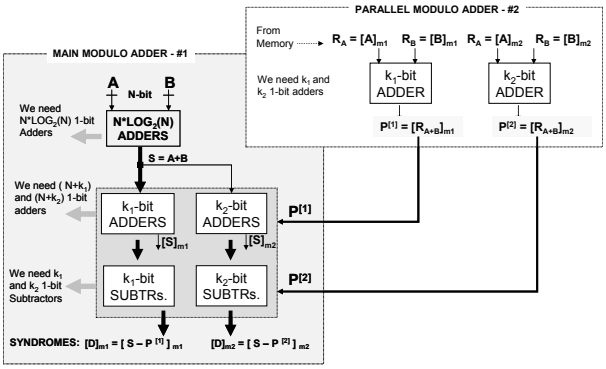

Figure 1. 1-bit ED and EDAC applied to an N-bit adder using 2 moduli.

Example Modulus Calculation: For a given modulus m and a simple 2-input 1-output operation with operands A, B and output S, we compute

$\mathbf{[A]_m}$, $\mathbf{[B]_m}$ and $\mathbf{[Q]_m}$

We have for an arbitrary operation *OP*:

(1) $\mathbf{[A]_m}$ *OP* $\mathbf{[B]_m = [Q]_m}$

If there is a 1-bit error then:

(2) $\mathbf{[A]_m}$ *OP* $\mathbf{[B]_m = [Q]_m \pm}$ *OP* $\mathbf{[2^i]_m}$

Where *i* indicates the position of the error reflected in the *Q* side of the operation. We can see from this example that it is possible to detect a single error with a single modulus. However, it is not possible to reliably find and correct the bit with the error as 2 or more *i* can be congruent modulo *m*. If we introduce a second modulus *n* however, and designate *r* and *s* as 2 *i* such that:

(3) $r \equiv s \bmod(m)$

then (3) implies that we can not have:

(4) $r \equiv s \bmod(n)$

for *n* and *m* that are relatively prime and each with magnitude in excess of the maximum range for *r* and *s* [5].

With the above conditions we can therefore find the position of a single bit error with two moduli.

Computations on residues are commutative, distributive and associative; therefore RAC can detect and correct for transients inside of the RAC augmented computation block using the reduction logic that is applied to the targeted operation.

RAC Design

In this work, two types of RAC designs are evaluated for both error detection (ED) and EDAC against a non-RAC augmented design. The designs are fundamental arithmetic modules consisting of an adder or multiplier with predetermined operand sizes. The initial adder or multiplier arithmetic block design is augmented with the full RAC-based EDAC for SEU faults. Figure 2 shows the components required for a non-optimized RAC augmentation of a Given Operation.

Figure 2 is an illustration showing how RAC with EDAC is implemented for 2 input operations. The *A* and *B* inputs as well as the *Q* output are registered (registers are not depicted in diagram). $M_{0A/B}$ are the residue computation blocks for reducing operands A & B modulo the first modulus M_0 [6]. $M_{0A/B}$ are the residue computation blocks for reducing operands A & B modulo the second modulus M_1. *OP* is the Combinatorial Logic Function/Operation that is RAC augmented. $OP_{M0/1}$ are the Operation blocks with operand sizes scaled to that of the moduli. $M_{0/1R}$ are the residue computation blocks for the outputs of $OP_{M0/1}$ blocks. $M_{0/1S}$ are the residue computation blocks for reducing the outputs of the *OP* blocks modulo the first modulus M_0 and the second modulus M_1. The $\Delta_{M0/1}$ blocks compute the difference in residue magnitude for each modulo between the Operation applied to the A & B operands and the Operation applied to the residues of the A & B operands. The $M_{0/1D}$ are the residue computation blocks for reducing the outputs of $\Delta_{M0/1}$ blocks modulo the first modulus M_0 and the second modulus M_1.

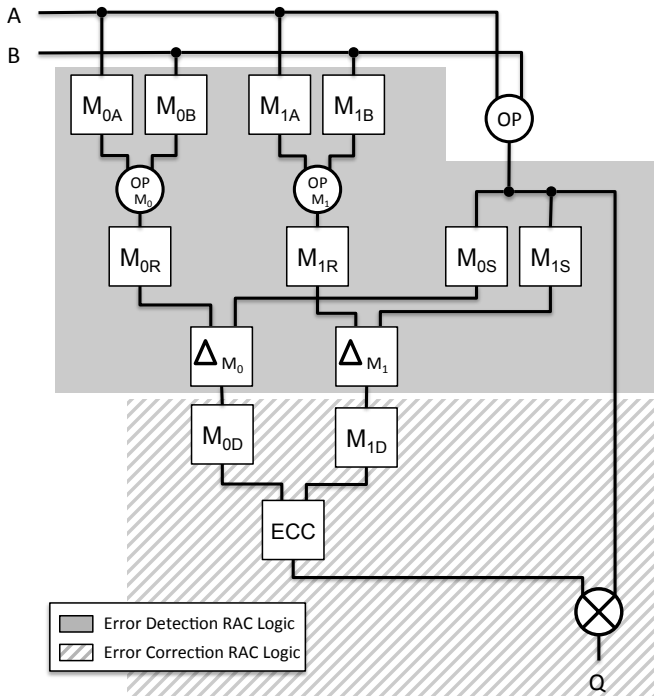

Figure 2. Logic Diagram of ED and EDAC sections of a RAC enhanced 2-input, 1-output operation.

Simulating the RAC Augmented Designs

Simulation Setup: A simple experiment is established to verify the functionality of the RAC design and to investigate the following: 1) The ability of a RAC enhanced circuit to detect single-bit errors, 2) The ability of a RAC enhanced circuit to correct single-bit errors. To stimulate these designs, fault injection hooks are added to all possible insertion points of the arithmetic blocks. A basic fault injector program with an exhaustive set of test patterns for single SEUs is developed and applied to the arithmetic blocks through the basic fault injector program. When subjected to single upsets, these systematic simulations give insight into the effectiveness, overhead, and performance for basic RAC enhanced elementary modules.

This initial study does not consider any physical placement, feature size or arithmetic module functional use patterns. The study is also independent of any specific fabrication technology or implementation platform. The Operation module that is to be augmented is described in HDL using generic operand sizes. All the RAC logic modules as well as the augmentation process are also described in generated HDL. The RAC augmented logic

test pattern generation, insertion and simulation for SEUs is produced and operated by the RAC augmentation process.

A node is defined as any input or output to a logic gate. For instance, a 3 input NAND gate has 4 nodes, 3 input nodes and 1 output node. For the purpose of this simple experiment, a fault injection epoch is defined as an arbitrary number between 2 and 10 system clock periods. All fault injection epochs are synchronous with the system clock. For each epoch, a single node is selected at random. The adder processes approximately 47.5 million bytes of arbitrary data during the experiment. Results are shown for two instances of a non-optimized RAC enhanced simple 8-bit adder were subjected to 1,000,000 single node flips.

Given that the injected bit flips are pseudo-random and are not correlated to targeted stages of an Operation nor to specific operand sets, not all single node-flips result in faults that would be latched in the Operation output registers set. Node-flips that do result in faults are designated as *Faults Injected.*

A *Fault Detected* is defined as an instance where the result produced by the plain Operation logic does not match the expected golden result AND the RAC logic generates a non-zero flag value. Non-zero RAC flag values indicate that the RAC logic estimates that an error has occurred either in the Operation Logic or within the RAC logic. Faults that affect the logic that generates the RAC flags will go undetected and uncorrected by RAC.

Faults requiring correction (FRC) are defined as faults that occurred in the Operation logic that require error correction, FRCs occur when the golden value and the non-RAC corrected value of an Operation for a given set of operands differ. The RAC logic does not successfully detect all the FRCs; the portion of faults that are caught and accurately corrected for are labeled FCA for *Faults with Correction Applied*. Faults that are either not caught by the RAC logic, or are valid FCAs to which an erroneous correction is applied or altogether invalid FCAs (false positive) are considered *Missed*.

Fault Injection Results: Figure 3 depicts the first instance of the RAC enhanced adder. The RAC logic set in combinatorial logic results in a ~2.7x logic overhead compared to a non-RAC enhanced adder. Figure 4 depicts the second instance of the RAC enhanced adder; this design implements the RAC residue computation logic and error correction table in hardened lookup tables. By using hardened error correction lookup tables, there is a significant reduction in the logic overhead to ~1.5x compared to the previous design. The two designs are otherwise identical in terms of RAC control and differentiation logic.

The specific type of radiation hardened state element used for the residue lookup tables in Figure 4 is not relevant. The experiment is designed in a manner such that no logic state within the lookup tables is affected by the fault injection. For this experiment, the residue and ECC lookup tables are essentially nodes with $2x[\log_2(M_{0/1})]$ inputs and $[\log_2(M_{0/1})]$ outputs. These node inputs and outputs are subject to bit flips just like the ports on any other node.

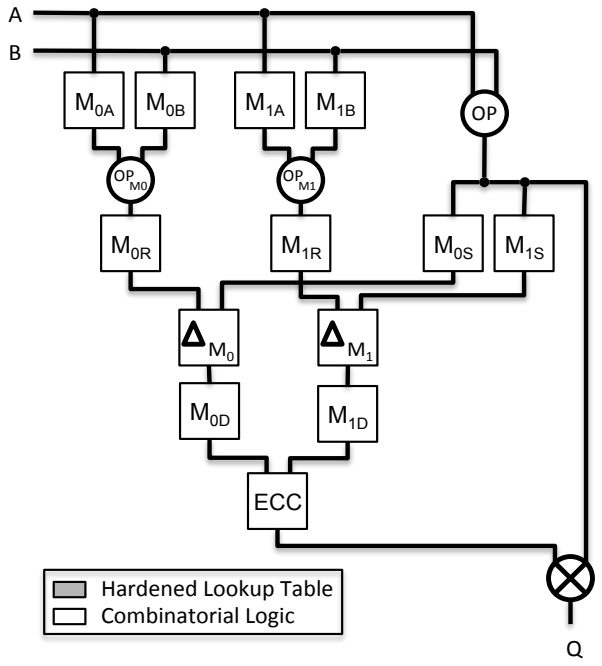

Figure 3. Logic Diagram of a RAC enhanced 2-input, 1-output operation with RAC circuitry entirely implemented in combinatorial logic.

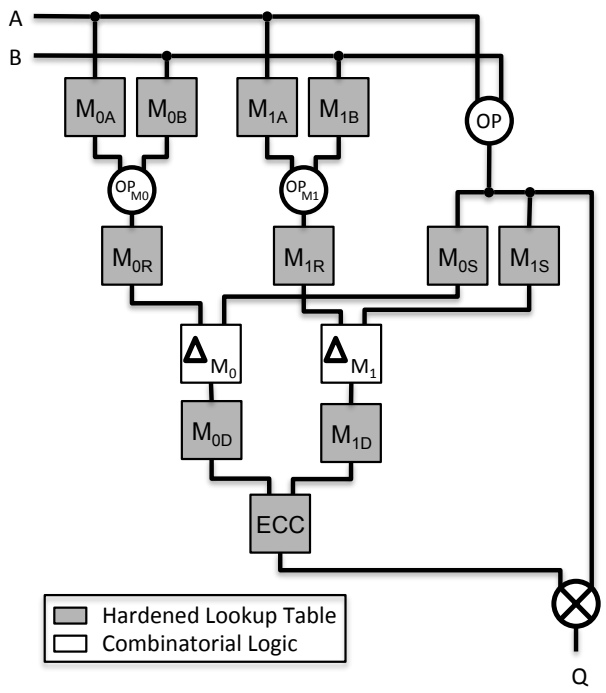

Figure 4. Logic Diagram of a RAC enhanced 2-input, 1-output operation with RAC residue circuitry implemented in combinatorial Radiation Hardened Lookup Tables

The resiliency results for the implementation in Figure 3 are summarized in Table 1.

Full RAC in Combinatorial Logic		
Metric	**Count**	**Rate**
Faults Injected	90,242	NA
Faults Detected	90,026	99.76%
FRC	52,884	58.74%
FCA	43,305	81.87%
Missed	216	0.24%

Table 1. Resiliency of full-combinatorial RAC augmented 8-bit general-purpose full adder when subjected to 1,000,000 single node flips and processing 47.5 million bytes of data. *Faults Injected* represents the number of single bit errors that would be latched in output registers without RAC. *Faults Detected* is the number of errors correctly detected by the RAC logic. *FRC* is the number of faults correctly determined by RAC as needing error corrections. *FCA* is the number of faults requiring correction that RAC successfully fixed. *Missed* is the number of faults that are neither detected nor corrected by RAC.

The resiliency results for the implementation in Figure 4 are summarized in Table 2.

Partial RAC in Combinatorial Logic		
Metric	**Count**	**Rate**
Faults Injected	66,066	NA
Faults Detected	65,984	99.88%
FRC	39,304	59.57%
FCA	29,873	76.00%
Missed	82	0.12%

Table 2. Resiliency of partial-combinatorial RAC augmented 8-bit general-purpose full adder when subjected 1,000,000 single node flips and processing 47.5 million bytes of data. *Faults Injected* represents the number of single bit errors that would be latched in output registers without RAC. *Faults Detected* is the number of errors correctly detected by the RAC logic. *FRC* is the number of faults correctly determined by RAC as needing error corrections. *FCA* is the number of faults requiring correction that RAC successfully fixed. *Missed* is the number of faults that are neither detected nor corrected by RAC.

The results show that for appropriately sized moduli, the RAC logic detects more than 99% injected faults that would result in an incorrect value being latched at the Q output.

Given that a relatively small amount of logic is required for the fault indication and flag generation circuitry, faults in this portion of the RAC logic are a rare occurrence (< 0.25%); RAC can not however correct for these faults.. RAC only accurately determines that 59% to 60% of FRC as actually needing correction. Of theses faults determined as needing correction, RAC accurately corrects 76% to 82% of errors. Given that RAC is essentially a reduction logic function, where a relatively large set of combined inputs are reduced to a small output, RAC ED is very sensitive to any faults that target any RAC enhanced computation modules. A side effect of this sensitivity is the limited ability of this non-optimized RAC implementation to perform accurate error correction. The partial-combinatorial RAC augmented circuit greatly reduces the logic overhead requirements compared to the non-optimized fully combinatorial implementation. The fault detection rate for the reduced overhead circuit is only slightly improved whereas the accuracy of fault corrections is deteriorated by almost 6 percentage points.

Conclusions

By performing aggressive SEU fault injections simulations for single bit-transients we have shown that RAC looks to be an attractive, lower overhead, alternative to triple modular redundancy for SEU hardening of logic operations. RAC is capable of detecting over 99% of

faults with as little as 50% overhead for a non-optimized implementation.

Acknowledgements

The authors would like to acknowledge the DTRA–CORRECT program for their financial support. (Contract #: HDTRA1-12-C-0057).

References:

[1] Niranjan, S., Frenzel, J.F. "A comparison of fault-tolerant state machine architectures for space-borne electronics", IEEE Transactions on Reliability, Volume 45, Issue 1 p 109-113, 1996

[2] H. Krishna, K. Y. Lin, and J. D. Sun, "A coding theory approach to error control in redundant residue number systems. Part I: Theory and single error correction," IEEE Trans. Circuits Syst., vol. 39, pp. 8-17, Jan. 1992.

[3] Daniel Lipetz, Eric Schwarz, "Self Checking in Current Floating-Point Units", Computer Arithmetic (ARITH), 2011 20th IEEE Symposium on Digital Object Identifier, 2011

[4] James L. Massey "Survey of residue coding for arithmetic errors" , Int. Computation Cent. Bull. (UNESCO Rome, Italy), vol. 3, no. 4, pp. 3-17, Oct. 1964.

[5] Stanislaw J. Piestrak "Self-Testing Checkers for Arithmetic Codes with Any Check Base A", Fault Tolerant Systems, 1991. Proceedings., Pacific Rim International Symposium on