Using Benchmarks for Radiation Testing of Microprocessors and FPGAs

Heather Quinn, William H. Robinson, Paolo Rech, Miguel Aguirre, Arno Barnard, Marco Desogus, Luis Entrena, Mario Garcia-Valderas, Steven M. Guertin, David Kaeli, Fernanda Lima Kastensmidt, Bradley T. Kiddie, Antonio Sanchez-Clemente, Matteo Sonza Reorda, Luca Sterpone, Michael Wirthlin

***Abstract*—Performance benchmarks have been used over the years to compare different systems. These benchmarks can be useful for researchers trying to determine how changes to the technology, architecture, or compiler affect the system's performance. No such standard exists for systems deployed into high radiation environments, making it difficult to assess whether changes in the fabrication process, circuitry, architecture, or software affect reliability or radiation sensitivity. In this paper, we propose a benchmark suite for high-reliability systems that is designed for field-programmable gate arrays (FPGAs) and microprocessors. We describe the development process and report neutron test data for the hardware and software benchmarks.**

***Index Terms*—soft errors, soft error rates, software fault tolerance, FPGAs**

I. Introduction

Well-crafted performance benchmarks play an essential role in computer architecture research [1]. These benchmarks are used to compare different architectures and evaluate different performance trade-offs. Benchmarks are also used to evaluate compiler technology and determine the effect of the compiler on the overall performance of the microprocessor. The great advances in microprocessor performance over the last few decades are due in part to the availability of high-quality performance benchmarks that represent a variety of real-world computing operations.

No such benchmarks are available for testing systems used in high radiation environments. This situation has developed because radiation-hardness assurance techniques are traditionally applied only to circuit layouts or manufacturing processes. Even though there are no standard test circuits available for these cases, most researchers compare their results to similar circuits, such as flip-flops or D-latches [2]–[4], so the community has found methods to work around not having standard benchmarks.

In recent years, the radiation effects community has grown to include organizations that are interested in testing and using complex, commercially available devices such as field-programmable gate arrays (FPGAs) [5] and microprocessors [6] in spacecraft or aircraft; and using parallel processors in large High Performance Computing (HPC) centers [7]. Many of these organizations are designing post-manufacturing hardware and software hardening techniques, such as modular redundancy, to increase resilience to single-event upsets (SEUs) and single-event transients (SETs). It is necessary to quantify how these techniques improve the resilience of these complex systems to single-event effects (SEEs). For these researchers, a standard benchmark of test circuits and codes would be beneficial.

A standard benchmark would define a set of codes/circuits and input vectors for testing that would ideally cover a number of realistic algorithms used in these components. Furthermore, information about the compilation/synthesis process and runtime environment provide a basis for repeatable results and provides standards for other researchers upon which to build. A baseline cross section without mitigation provides a basis for determining whether mitigation methods effectively mask SEEs and allows mitigation methods to be compared for power, effectiveness, and overhead.

Currently, without a benchmark researchers have been testing:

1) Homemade, synthetic designs that represent worst-case scenarios,
2) Circuits from OpenCores [8], existing benchmarks, or a vendor's intellectual property (IP) generation tool, or
3) Proprietary designs that have been used previously by the researchers and thus not available to other researchers to verify and compare results.

These approaches make it difficult to assess the relative radiation/reliability sensitivities, as no two organizations are using the same set of circuits or codes. It also causes researchers to focus on the parts of the test that can be compared, such as bit (or device) cross sections or failure in time (FIT) rates[1]. Even these representations can be difficult for comparing results

Heather Quinn, Los Alamos National Laboratory, ISR-3 Space Data Systems, Los Alamos, NM, 87545 USA, e-mail: hquinn@lanl.gov

W. H. Robinson and B. T. Kiddie are with Vanderbilt University, Department of Electrical Engineering and Computer Science, Nashville, TN, 37235, USA

P. Rech and F. Kastensmidt are with the Instituto de Informatica, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brasil

A. Barnard is with Stellenbosch University, Stellenbosch, South Africa

M. Aguirre is with the Universidad de Sevilla, Sevilla, Spain

M. Desogus, M. Reorda and L. Sterpone are with Politecnico di Torino, 10129 Torino (TO), Italy

L. Entrena, A. Sanchez-Clemente and M. Garcia-Valderas are with Universidad Carlos III de Madrid, Madrid, Spain

S. M. Guertin is with the Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, 91109, USA

D. Kaeli is with Northeastern University, Electrical and Computer Engineering, Boston, MA, 02115, USA

M. Wirthlin is with the NSF Center for High-Performance Reconfigurable Computing (CHREC), Department of Electrical and Computer Engineering, Brigham Young University, Provo, UT 84602, USA

[1]FIT is defined as the number of failures that can be expected in one billion (10^9) device-hours of operation.

across organizations, if the same set of circuits and codes are not used.

The reliability-focused benchmark suite described in this paper aims to provide radiation effects and reliability researchers with a consistent basis for current and future research. The suite would allow designers and researchers to:

- Assess relative reliability improvements between mitigated and unmitigated designs;
- Compare mitigation methods for improvements in reliability, performance, area, and power;
- Compare algorithms across architectures and process changes;
- Assess architectural effects on reliability, such as using or not using caches;
- Assess the effect of coding methods on reliability, such as iterative solvers; and
- Compare test methodologies across organizations, including both radiation testing and fault injection.

To this end, ten different organizations have collaborated since March 2014 to develop a suite of benchmarks for high-reliability applications. This paper presents proposed benchmarks for mitigated codes and circuits. We present data collected from testing FPGAs, microcontrollers, and parallel processors using the proposed benchmarks.

The paper is organized as follows. A survey of related benchmarks are presented in Section II. We present our rationale for radiation benchmarks in Section III. Information about the hardware and software benchmarks is presented in Sections IV and V. Experimental data from neutron testing are presented in Section VI. Finally, Section VII summarizes the paper.

II. Related Work

In this section, we list the benchmarks that helped compiler designers and computer architects to quantify how their changes affect standard architectures. We continue with a discussion of benchmarks used to assess software performance and testability. We then present hardware benchmarks that have been useful in design for test (DFT), where the benchmarks allow researchers to explore testability and automatic test pattern generation (ATPG). We start by discussing SEE characterization testing, due to its similarities to benchmark testing.

A. Standards for SEE Characterization testing

SEE characterization testing focuses on measuring the SEE cross sections for components. For example, SEE characterization tests for a microprocessor would measure the SEU sensitivities for caches and registers; SET sensitivities for the logic; and SEFI sensitivities for the control logic. SEE characterization tests for an FPGA would measure the SEU sensitivities for the configuration memory, user memory and flip-flops; SET sensitivities for the clocks and logic blocks; and the SEFI sensitivities for the control logic and reconfiguration ports. The standards for SEE characterization tests for microprocessors and FPGAs are still evolving. There are many fundamental questions that are unanswered. The benchmark we are proposing is not intended for basic SEE characterization tests. The assumption for this benchmark is that basic SEE characterization has already been completed. It is not within the scope of this paper or this research to clarify standards for SEE characterization tests.

For researchers looking for existing SEE characterization test guidelines, there are a number of different documents. Irom described a guideline [9] has been the standard for single-core microprocessors for several years. The follow-on document from Guertin [10] is useful for understanding how to perform multi-core microprocessor testing. Berg [11] provides information about FPGA testing. Quinn [12] provides guidance for testing complex systems.

B. Software Benchmarks

Two of the original software benchmarks are Dhrystone and Whetstone, which are both synthetic benchmarks [13], [14] created to represent the profile and behavior found in real applications. Modern compilers have been able to undermine the benchmarking process through Dhrystone [15]. Whetstone and Dhrystone have been replaced with more sophisticated benchmarks, such as SPECint/SPECfp, Linpack, and CoreMark. The Standard Performance Evaluation Corporation (SPEC) was established as an independent organization to maintain software benchmarks [16]. The SPEC benchmarks cover both integer and floating point operations. Both benchmarks are maintained with regular updates to the benchmark to keep current with modern computing standards and applications. This maintenance includes regular updates about problems with the benchmark that affect the benchmarking process [17]. Linpack is most commonly used to compare supercomputer ratings [18]. CoreMark is maintained by the Embedded Microprocessor Benchmark Consortium (EEMBC) [19]. CoreMark is meant as a replacement for Dhrystone, especially for smaller microprocessors [15]. Unlike Dhrystone, CoreMark has specific rules for implementing the benchmark to maintain the integrity of the benchmark.

Lately, some benchmark suites have been developed to evaluate the efficiency of parallel or heterogeneous HPC machines. These benchmarks are continuously improving. Examples of such suites are the University of Virginia RODINIA suite [20], the NASA NAS Parallel Benchmark [21] and the Sandia mini-applications [22]. Both RODINIA and NAS provide a representative set of HPC algorithms written in portable languages. The suite includes algorithms that are both memory-bound and compute-bound so that an HPC architecture's performances can be measured under typical scenarios.

The final set of benchmarks presented are two benchmarks designed for assessing power/energy efficiency. One benchmark was used by the Defense Advanced Research Projects Agency (DARPA) Power Efficiency Revolution for Embedded Computing (PERFECT) program as part of the test, assessment, and verification effort that was led by the Pacific Northwest National Laboratory (PNNL) [23]. The PERFECT benchmark suite includes both applications and code kernels; it covers a range of application types. There is also the EEMBC benchmark for power, called the EnergyBench [24].

This benchmark is designed to assess power consumption while running the EEMBC performance benchmarks. Like the EEMBC performance benchmarks, the EnergyBench benchmark includes methods for certifying the results.

C. Hardware Benchmarks

Several hardware benchmarks were specifically designed by the DFT and ATPG communities, such as ISCAS 85/89, ITC'99, and IWLS 2005. The ISCAS 85 benchmark has ten combinational circuits that were distributed in netlist format [25], [26]. It has since been translated into VHDL (VHSIC Hardware Description Language) and Verilog [27] for ease of implementation. The benchmark was extended to include sequential circuits as well [28]. The ISCAS 85/89 benchmarks are not packaged with expected stimuli. The ITC'99 benchmark has many different flavors; most of which were implemented in VHDL, and some of which were implemented in Electronic Design Interchange Format (EDIF). The version of ITC'99 in which we are interested is the I99T benchmark that was implemented by Politecnico di Torino [29], [30]. Finally, there is the IWLS 2005 benchmark [31] that was designed by Christoph Albrecht from the Cadence Research Laboratories at Berkeley. The IWLS 2005 benchmark includes 84 circuits that were collected from various sources, including ISCAS 85/89, ITC'99, and OpenCores. The IWLS 2005 benchmark is available in Verilog.

III. Rationale for Using Benchmarks for Radiation Testing

There are several criteria that we use when evaluating and developing benchmarks, including:

- Repeatability of benchmark tests,
- Representative of deployed computing workloads,
- Availability of fixed input vectors, and
- Cross-platform implementation.

The ability to repeat test results is an important part of standardized testing. By reporting the algorithms, the input vector sets, the compilation/synthesis settings, and the runtime environments, enough information about tests should be defined so that researchers can either repeat or build from published results. We will discuss our rationale behind the need for standard algorithms, input vectors and portability in a benchmark in this section.

We believe that a benchmark of realistic algorithms is more likely to be adopted by the radiation effects community, because test results will provide realistic insight into spacecraft. Furthermore, providing a defined set of algorithms to use for testing will provide a basis for many organizations to compare their results. By providing several different types algorithms, it is possible that different aspects of the mitigation methods can be highlighted. This variety allows researchers to create focused mitigation methods that work well a subset of algorithms. Matrix operations are very heavily used in HPC algorithms, which has led to mitigation methods that focus only on protecting matrices [32]–[34]. While these matrix mitigation methods might not perform well for the entire benchmark, they should out-perform most general mitigation methods for matrix algorithms. Therefore, it is necessary to provide a wide variety of algorithms that allows a wide range of mitigation techniques to be tested.

TABLE I
Size and functionality of I99T Suite circuits

	VHDL		Structures		Function
	Lines	Proc	Gates	FF	
b1	110	1	49	5	FSM
b2	70	1	27	4	FSM
b3	141	1	153	30	Arbiter
b4	102	1	628	66	Min/max
b5	332	3	574	34	Memory
b6	128	1	55	9	IRQ
b7	92	1	427	49	Count
b8	89	1	171	21	Find inclusions
b9	103	1	160	28	S2S Converter
b10	167	1	180	17	Voting
b11	118	1	548	31	Cipher
b12	569	4	1006	121	Game
b13	296	5	317	53	Analog sensor
b14	509	1	5678	245	Viper
b15	671	3	7974	448	80386

For circuit tests, using defined input vector sets is essential in creating repeatable experiments. Many hardware errors can only be observed when the circuit is exercised with specific inputs. The DFT and ATPG communities have studied methods for designing optimal input vector sets to cover particular types of errors, such as stuck at faults, which can be useful for our efforts. It is an open question whether the input vector values affect the sensitivity of software, although there are several known instances of where input vector size affects software cross sections [35]–[37]. Furthermore, there is no software equivalent to the DFT and ATPG communities, so the effect of input vectors on software is not as well understood as hardware. While it remains an open question, we would prefer to provide input vector sets to the software benchmark to at least focus on repeatability.

Finally, implementing algorithms in portable languages will allow the benchmark to be implemented on several platforms. Adoption of the benchmarks will be more likely if the circuits/codes can be built for many different platforms with a wide variety of tools. To that end, algorithms that are implemented in platform-specific languages, such as Native Circuit Description (NCD) for Xilinx FPGAs or assembly language for microprocessors, limits the ability to compare cross sections across different architectures. It is also ideal to provide the algorithms in source code, so that test designers can compile the codes or synthesize the circuits using the set of tools they expect to use on the deployed system. To meet these requirements, we look for scalable circuits in standard hardware description languages (HDLs) or scalable software in standard programming languages that represented a wide range of functionality.

Although an initial set of benchmarks is presented in this paper, we anticipate that this benchmark suite will expand and evolve over time. As we gain experience with this initial set of benchmarks, we anticipate expanding the benchmark suite to include other benchmarks that address the limitations of the current benchmarks and test additional features or capabilities. Like most microprocessor benchmark suites, a variety of benchmarks are needed to fully test the features of a microprocessor or FPGA architecture.

IV. FPGA Radiation Benchmark

The FPGA benchmark leverages ITC'99, which is a well-established ATPG benchmark. This benchmark meets all of our requirements, including a variety of realistic algorithms, defined inputs, scalability, and portability. This benchmark is specifically designed for testing and includes a set of input vectors that are designed by the ATPG community. The circuits are implemented in a common HDL, which makes them easy to implement on different types of FPGAs. Because many of the circuits are small, it is possible to synthesize the circuits for a variety of differently sized FPGAs, even if the mitigation technique increases the size of the circuit. Finally, these circuits implement a wide range of functions from finite state machines (FSM) to soft-core microprocessors. Many of the circuits are realistic to how FPGAs are used in embedded computing environments. Because of all the qualities described, the ITC'99 benchmark is ideal for our testing needs.

We are specifically adopting the I99T portion of the benchmark. The first 15 circuits in this benchmark are listed in Table I; the rest are combinations and implementations of the listed circuits. We initially focused on B13 at this stage and collected data on this circuit in a radiation environment.

V. Software Radiation Benchmark

The software benchmark has been harder to design than the FPGA benchmark. We were originally hampered by trying to develop a benchmark that would have a standard set of algorithms that would work with a range of processing architectures, including 16-bit microcontrollers, multi-core microprocessors, and high-end Graphics Processing Units (GPUs). While we could find individual codes, such as EEMBC's CoreMark, that could work on nearly all architectures, it is not useful to compare an embedded microprocessor to a GPU. Furthermore, existing software benchmarks tend to bundle all of the functionality into one code. Therefore, there is not an equivalent of I99T for software with multiple codes covering different types of program coding styles and realistic workloads. To that end, we are designing our own benchmark suite of scalable algorithms with specific implementations for classes of microprocessors.

The proposed software benchmark includes both a collection of different, simple algorithms that are realistic software codes, and a few standard software benchmarks. We specifically want to cover sorting, Fast Fourier Transforms (FFTs), and matrix algorithms, as they are commonly used in many applications and are useful for evaluating the reliability of parallel processors [38]–[40]. We also want to represent different data and program structures, because the corresponding implementation of mitigation methods may differ as well. We are also interested in algorithms where the computational load could be homogeneously divided into parallel processes or run on a single core. We are interested in leveraging some of the existing benchmark suites.

The algorithms that we implemented for the software benchmark are:

- Advanced Encryption Standard (AES) 128,
- Cache test,
- CoreMark,
- Fast Fourier Transform (FFT),
- Hotspot,
- HPCCG,
- Matrix multiply (MxM), and
- Quicksort (Qsort)

Information about these codes can be found in Table II. Table II also indicates the two algorithms that are synthetic and are specifically designed for test purposes. In some cases, multiple implementations for parallel processors are available, where trade-offs can be made between memory latencies and performance. In the benchmark suite, we include both a memory-bound and a computing-bound version of MxM to account for the effects of algorithm implementation. All of the algorithms are implemented in C, C++, and/or CUDA to maximize implementation across several architectures.

Since most of the codes are scalable to microprocessors with differing amounts of memory, by separating microprocessors into classes, we can specify the problem size used for the algorithm. We organize microprocessors into one of these standard classes:

- General-purpose, multiple-core microprocessors,
- General-purpose, single-core microprocessors,
- Microcontrollers and ARM cores, and
- Soft-core microprocessors.

These classes each have a version of the benchmark that could be implemented without changes to the functionality or memory usage. These classes allow a standard set of rules for executing the benchmark so that the results cannot be "gamed." By adjusting the benchmark and the rules for each class, test results are standardized and easier to compare. Furthermore, it will keep researchers from comparing dissimilar hardware.

We are still determining the role of input vectors in this benchmark suite. Five out of the eight test codes have defined input test vectors, while three (i.e., Fast Fourier Transform, MxM, and Qsort) use randomized test vectors. While randomized, MxM and Qsort use the C library for random with specific seeds so that the test vectors are repeatable. Now that a baseline of test results for the software benchmark has been determined, it is possible that focused testing on input vector values and sizes can be completed to define the input vectors for the three algorithms. In particular, we need to focus on Qsort, as that is the only algorithm that makes control flow changes based on data, whereas FFT and MxM only compute with the data.

TABLE II
SOFTWARE BENCHMARK CHARACTERISTICS INCLUDING THE USE OF ARRAYS (ARR) OR LINKED LISTS (LIST) AS DATA STRUCTURES. OTHER QUALITIES INCLUDE WHETHER THE CODE IS SYNTHETIC, INCLUDES MATRIX OPERATIONS, OR IS ALREADY USED AS A PERFORMANCE BENCHMARK.

	Data Structure		Program Structure		Other Qualities		
	Array	List	Recursion	Iteration	Synthetic	Matrix Ops	Performance
AES							
Cache	X				X		
Coremark		X			X	X	X
FFT							X
Hotspot							X
HPCCG	X			X		X	X
MxM	X					X	
Qsort	X		X				

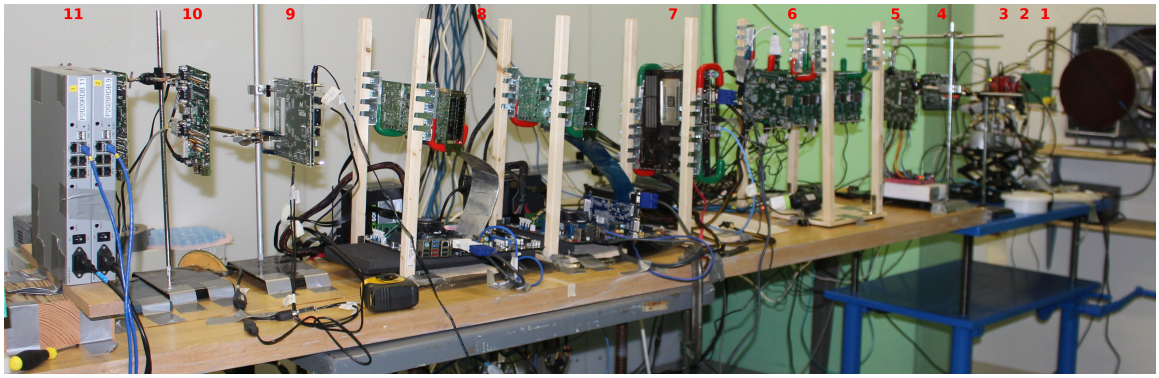

Fig. 1. Picture of LANSCE Test. The numbers at the top of the list correspond to the tests shown in Table III.

TABLE III
EXPERIMENTS IN THE LANSCE TEST. †MITIGATED AND UNMITIGATED BENCHMARK, *ONLY UNMITIGATED

N.	Benchmark	Organization	Component
1	s/w†	LANL	TI MSP430F2619
2	s/w†	LANL	TI MSP430FR5739
3	s/w†	LANL	TI Tiva
4	h/w†	Madrid	Artix-7
5	h/w†	Torino	Virtex-5
6	s/w†	UFRGS	6 x Zynq
7	s/w*	UFRGS	Kaveri A10 APU
8	s/w†	UFRGS	Tesla K20 GPUs, Xeon-Phi
9	h/w*	UFRGS	Virtex-5
10	s/w*	JPL	Atmel AT91SAM9G20
11	s/w*	JPL	Freescale P2020
12	s/w + h/w†	BYU	Kintex-7
13	s/w*	BYU	Zynq
14	custom*	Vanderbilt	

VI. RADIATION TESTS USING THE BENCHMARKS

Initial radiation tests were completed at the Los Alamos Neutron Science Center (LANSCE) in December 2014. Fig. 1 shows the test setup at LANSCE, although several boards were added to the test later. The tests represented are shown in Table III. A follow-on test was completed in January 2015. The results of these tests were analyzed to determine cross sections using the number of the errors from the tests and the fluence from LANSCE's dosimetry. The cross sections were then converted to FITs using the fast neutron flux for New York City (13 neutrons/cm^2/hour).

The following discussion will provide results for micro-controllers, ARM cores, GPUs, and FPGAs. These results show the sensitivity to corrupted calculations, which is called Silent Data Corruption (SDC) [41], and single-event functional interrupts (SEFIs) in terms of FITs. For both SDC and SEFIs, the FIT is reported, which is normalized to the FIT of the unhardened version, called Relative FIT ($R.FIT$), to allow a quick comparison between different hardening strategies applied to different codes or devices. The overhead represents either: (1) the increase in area for hardware, or (2) the increase in execution time for software. While it is possible that hardware mitigation could cause an increase in the execution time, it would only happen if it is necessary to decrease the clock to meet timing.

A. Hardware Benchmark Results

For the circuit benchmark, we focused on the radiation testing of the B13 circuit from ITC'99 benchmark suite. This circuit is very small, so it was replicated 30 times for each of the implementations on a Virtex-5LX50T Xilinx SRAM-based FPGA. At LANSCE we tested an unmitigated version, an X-

TMR mitigated version and a X-TMR/VERI-Place mitigated version. The circuits were synthesized using the 9.2i version of the Xilinx Integrated Synthesis Environment (ISE) Design Suite. This version of ISE can be integrated with the X-TMR tool. The X-TMR designs mitigated all of the logic. None of the global signals (input, output, clock or reset) are mitigated. These signals are registered immediately, which means that the signals will triplicate after the first latch. We have found that this method of handling global signals provides a reasonably low cross section without the effort of making timing using three sets of signals. The VERI-Place versions uses POLITO's software mitigation tool [42]. The X-TMR and VERI-Place implementation take the same amount of FPGA resources, because the VERI-Place mitigation tool hardens the circuit's physical netlist by acting on the logic placement position. The test fixture does not scrub SEUs as SEUs accumulate, but once the circuit reports an error. This means that there can be one to many SEUs that contribute to an SEU.

Table IV reports the results for the three implementations. The results report SDCs from the mitigated circuits normalized to the SDCs from the unmitigated circuits, while the overhead represents the increase of the circuit area. In these results, the X-TMR results are larger than one would expect. One issue with allowing SEUs to accumulate is that the X-TMR mitigated circuits are only guaranteed to mask one SEU. Once more than one SEU accumulates, a mitigated circuit is likely to fail at three times the rate of the unmitigated circuit, because of the increased size of the circuit from the mitigation process. Therefore, it is possible that the cross section for a mitigated circuit can be three times larger than an unmitigated circuit when SEUs accumulate. These results show how well X-TMR-mitigated circuits can mask accumulating SEUs. The VERI-place-mitigated circuits perform even better than the X-TMR-mitigated circuits under these conditions.

We need to continue experimentation with the hardware benchmark, including testing more circuits and testing more architectures. Although we focused on B13 for now, there are many circuits available in the I99T. Fault emulation can be helpful in testing more circuits in a uniform and thorough manner, as fault emulation tests are less expensive than radiation tests. We are preparing more circuits for radiation tests by using fault emulation using the FT-UNSHADES platform. FT-UNSHADES is able to emulate SEUs in both configuration memory and the user flip-flops [43]–[45]. A diagram of the system from [44] is shown in Fig. 2. One of the advantages of FT-UNSHADES is that it is designed to load input test vectors from DRAM, making it straightforward to work with automated test vectors. The ability to test both the configuration memory and the user flip-flops also increases coverage of the circuit before radiation testing. Because radiation testing could only cover a small portion of the testing, we intend to use FT-UNSHADES to provide insight into the other circuits through the use of fault emulation.

Finally, many new types of FPGA architectures are newly accessible to the space community, such as the Altera Stratix-V and the Microsemi RTG4. We would like to test more architectures so that we could provide a cross-architecture comparison.

TABLE IV
B13 TEST RESULTS ON THE XILINX VIRTEX-5 LX50T WITH 95% CONFIDENCE INTERVALS

Impl.	SDC		Overhead
	FIT	R. FIT	
Unhardened	$(2.10 \pm 0.03) \times 10^3$	1.0	1.0
X-TMR	$(1.72 \pm 0.01) \times 10^3$	0.82	4.56
VERI-Place	$(1.34 \pm 0.04) \times 10^2$	0.06	4.56

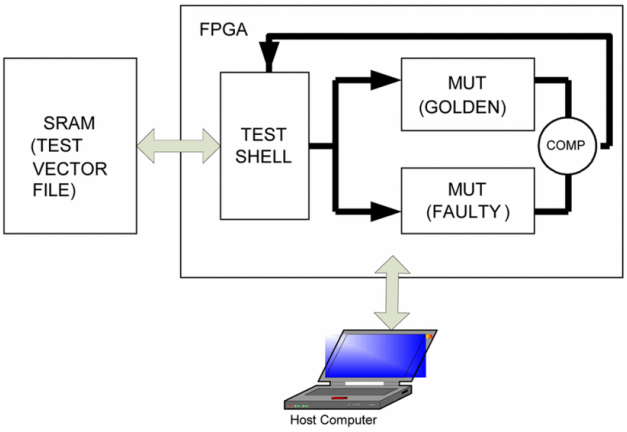

Fig. 2. FT-UNSHADES system for fault injection of a Module Under Test (MUT) [44].

B. Software Benchmark Results

We completed neutron testing of the software benchmark on a number of components, including a flash-based microcontroller, a ferroelectric-memory-based microcontroller, two ARMs, and two GPUs. Many of these components were tested with both mitigated and unmitigated codes.

Table V show the results for all of tested codes in the software benchmark for two different microcontrollers and two ARMs. The Texas Instruments components used the Code Composer Studio compiler version 6.0. The software for the two MSP430s was compiled with register optimizations (O0), and the software for the Tiva was compiled with global optimizations (O2). The Texas Instruments components do not have a runtime environment. The Zynq ARM used the Xilinx 14.4 Software Development Kit compiler with register optimizations (O0). We use the "bare metal" runtime environment for the Zynq ARM. Each of the microcontroller and ARM codes are mitigated using LANL's Trikaya software mitigation technique, which employs spatial and temporal triple modular redundancy (TMR) [37].

Because each of these microprocessors has only a small amount of SRAM, the FIT rates are very small. In some cases, there are no errors from the code during many days of testing. In these cases, the FIT is based on $fluence^{-1}$, and the Poisson confidence interval for null data is used. While most of the time the mitigation decreases the FIT for the software, there are few cases where that is not true. In two of these cases, the mitigated code had no failures, and FIT based on $fluence^{-1}$ is larger than the unmitigated FIT that was tested longer. In the MSP430F2619 Qsort tests, one run of the software had 23

TABLE V
SOFTWARE BENCHMARK RESULTS FOR MICROCONTROLLERS AND ARM CORES IN FITS WITH 95% CONFIDENCE INTERVALS

		TI MSP430F2619	TI MSP430FR5739	Xilinx XC020 ARM	TI Tiva ARM
AES	Unmitigated	0.38 (0.04, 1.37)	0.85 (0, 3.1)	11,711 $\pm$ 203	0.30 (0, 1.1)
	Mitigated	3 (1,5)	2 (0,7)	1 (0,4)	0.31 (0, 1.1)
$Cache$	Unmitigated	8 $\pm$ 2	10 (6, 15)	12 (6, 22)	75 $\pm$ 10
	Mitigated	0.21 (0, 0.76)	2 (0, 8)	1 (1, 4)	0.27 (0, 1.0)
$CoreMark$	Unmitigated	1.27 (0.51, 2.61)	N/A	1 (0, 4)	0.75 (0.15, 2.2)
MxM	Unmitigated	4 (2, 6)	1 (0, 4)	2 (0.2, 8)	59 $\pm$ 13
	Mitigated	0.27 (0, 1.0)	2 (0, 8)	1 (0.1, 6)	10 (7, 14)
$QSort$	Unmitigated	3 (2, 5)	25 (16, 38)	3 (0.7, 10)	59 $\pm$ 13
	Mitigated	0.23 (0,0.88)	–	1 (0.1, 6)	–

TABLE VI
SOFTWARE BENCHMARK RESULTS FOR NVIDIA K20 IN FITS WITH 95% CONFIDENCE INTERVALS

bench.	config	SDC		SEFI		Overhead
		FIT	R. FIT	FIT	R. FIT	
MxM	Unhardened	$(4.63 \pm 0.80) \times 10^2$	1.0	$(3.97 \pm 0.52) \times 10^2$	1.0	1.0
	ECC	44.91 ± 9.94	0.097	$(6.07 \pm 1.25) \times 10^2$	1.523	1.01
	ABFT	8.34 ± 0.96	0.018	$(4.47 \pm 0.92) \times 10^2$	1.128	1.14
FFT	Unhardened	$(2.88 \pm 0.39) \times 10^3$	1.0	$(7.02 \pm 0.86) \times 10^2$	1.0	1.0
	ECC	$(4.14 \pm 0.88) \times 10^2$	0.144	$(1.01 \pm 0.25) \times 10^3$	1.436	1.50
	ABFT	51.84 ± 6.67	0.018	$(8.20 \pm 0.87) \times 10^2$	1.145	1.18
$Hotspot$	Unhardened	$(2.04 \pm 0.31) \times 10^2$	1.0	$(1.12 \pm 0.17) \times 10^2$	1.0	1.0
	ECC	18.16 ± 2.01	0.089	$(1.11 \pm 0.91) \times 10^2$	1.439	1.00
	spacial DWC	3.26 ± 0.45	0.016	84 ± 0.92	0.750	2.45
	time DWC	2.45 ± 0.34	0.012	8.85 ± 0.87	0.079	1.90

values in two arrays corrupted simultaneously, which caused 72 errors in the output values. While 60 errors were corrected through the mitigation process, 12 were uncorrected. These 12 uncorrected errors skewed the results for that code as this one run is the only run of 52 total runs that had any uncorrected errors.

The AES code takes approximately the same amount of memory space in each microcontroller, and the differences in the cross sections are based on architectural differences among the microcontrollers. The exception is the unmitigated implementation on the Xilinx Zynq ARM, which had thousands of errors during the test. In this case, the test vectors were stored in SRAM, due to the component's lack of on-chip non-volatile memory. The mitigated version of the Zynq ARM AES code was able to suppress all of the SEUs in the test vectors.

The Trikaya technique is effective on many of the codes, masking all or nearly all SEUs. For MxM, over 2 million faults in the results matrix are masked using TMR, and only 41 faults are considered uncorrected. In comparison to other methods, the increase in overhead is much higher: 1.01–2.99 increase in data variables and 1.02–2.40 increase in instructions. The execution time increases about 1.34–3.29 times, although there was a catastrophic mitigation of a cache test code that increased the execution time by 983.22 times due to an overuse of *printf* code.

Table VI shows the results for software benchmarks on GPUs. We tested the 28-nm NVIDIA $Kepler$ K20 GPU. There are a total of 14 Streaming Multiprocessors (SMs) and 192 Compute Unified Device Architecture (CUDA) cores within each SM. Each SM has 64K registers, 64KB of combined shared memory and L1 cache, and 48KB of read-only data cache. SMs share 1536 KB of L2 cache and a total 6GB GDDR5 memory. Register files, shared-memory, L1 and L2 caches are SECDED protected, read-only data cache is parity protected. The K20 has CUDA capabilities 3.5 and can execute up to 192 parallel threads per SM in a single computing cycle. Tests were performed using a host PC running Ubuntu 14 and CUDA compiler 7.0.

We implemented several different mitigation methods on the NVIDIA K20 GPU: applying error-correcting codes (ECC) to caches, registers, and shared memory; an Algorithm-Based Fault Tolerance (ABFT) strategy for MxM and FFT; and spatial and temporal Duplication With Comparison (DWC) for Hotspot. These results show that the increase in overhead for ECC is very small, but the sensitivity to SEFIs increases, due to ECC failures with multiple-cell upsets (MCUs). From Table VI, it is easy to compare the impact of different hardening strategies for MxM. It seems that ECC on GPUs has a similar reliability improvement to LANL's Trikaya technique, but imposes a lower overhead. On the contrary, ABFT seems

extremely efficient compared to other techniques.

In the future, dynamic profiling of the software benchmarks and other codes would be helpful in determining the efficiency of the software benchmark. Dynamic profiling will help users to determine what percentage of time the code is spending loading/storing memory values, branching, or completing other operations. This type of information could help designers to develop benchmarks with a greater overlap with flight codes. Unfortunately, there is not an inexpensive method for obtaining dynamic profiling results over a wide range of microprocessors. Furthermore, most of the existing tools are concerned with test coverage, memory leaks, or security vulnerabilities. This topic is best handled in detail elsewhere.

VII. Summary

We have initiated a common set of hardware and software benchmarks to evaluate reliability and radiation effects for mitigation methods on FPGAs and microprocessors. These benchmarks were used in neutron radiation testing to demonstrate improvements in reliability for both FPGA circuits and processor executables. The improvements in reliability are achieved at an additional cost of hardware for FPGA circuits and lower performance for the processor executables. The use of these benchmarks will help to evaluate the trade-off between improvements in reliability and additional cost. These benchmarks can be useful for reliability and radiation effects researchers who are interested in both understanding the dynamic behavior of their systems and implementing mitigation methods.

References

[1] J. L. Henning, "SPEC CPU2000: measuring CPU performance in the new millennium," *Computer*, Vol. 33, No. 7, pp. 28–35, Jul 2000.

[2] T. Li, H. Yang, G. Cai, T. Zhi, and Y. Li, "A CMOS triple inter-locked latch for SEU insensitivity design," *IEEE Trans. Nucl. Sci.*, Vol. 61, No. 6, pp. 3265–3273, Dec 2014.

[3] K. Rodbell, D. Heidel, J. Pellish, P. Marshall, H. Tang, C. E. Murray, K. LaBel, M. Gordon, K. Stawiasz, J. Schwank, M. Berg, H. Kim, M. Friendlich, A. Phan, and C. Seidleck, "32 and 45 nm radiation-hardened-by-design (RHBD) SOI latches," *IEEE Trans. Nucl. Sci.*, Vol. 58, No. 6, pp. 2702–2710, Dec 2011.

[4] M. Glorieux, S. Clerc, G. Gasiot, J.-L. Autran, and P. Roche, "New D-flip-flop design in 65 nm CMOS for improved SEU and low power overhead at system level," *IEEE Trans. Nucl. Sci.*, Vol. 60, No. 6, pp. 4381–4386, Dec 2013.

[5] S. M. Trimberger, "Three ages of FPGAs: A retrospective on the first thirty years of FPGA technology," *Proceedings of the IEEE*, Vol. 103, No. 3, pp. 318–331, 2015.

[6] H. Quinn, T. Fairbanks, J. L. Tripp, G. Duran, and B. Lopez, "Single-event effects in low-cost, low-power microprocessors," in *IEEE Radiation Effects Data Workshop (REDW)*, 2014, pp. 1–9.

[7] M. Snir, R. W. Wisniewski, J. A. Abraham, S. V. Adve, S. Bagchi, P. Balaji, J. Belak, P. Bose, F. Cappello, B. Carlson, A. A. Chien, P. Coteus, N. A. DeBardeleben, P. C. Diniz, C. Engelmann, M. Erez, S. Fazzari, A. Geist, R. Gupta, F. Johnson, S. Krishnamoorthy, S. Leyffer, D. Liberty, S. Mitra, T. Munson, R. Schreiber, J. Stearley, and E. V. Hensbergen, "Addressing failures in exascale computing," *International Journal of High Performance Computing Applications*, Vol. 28, No. 2, pp. 129–173, 2014.

[8] "OpenCores website," Last accessed on 6/2015. [Online]. Available: http://opencores.org

[9] F. Irom, "Guideline for ground radiation testing of microprocessors in the space radiation environment," Jet Propulsion Laboratory, Tech. Rep. 13, 2008.

[10] S. M. Guertin, "A guideline for SEE testing of SOC," in *Single Event Effects Symposium, San Diego, CA*, May 2013.

[11] M. Berg, "Field programmable gate array (FPGA) single event effect (SEE) radiation testing," NASA/Goddard Space Flight Center, Tech. Rep., 2012.

[12] H. Quinn, "Challenges in testing complex systems," *IEEE Trans. Nucl. Sci.*, Vol. 61, No. 2, pp. 766–786, 2014.

[13] S. Harbaugh and J. A. Forakis, "Timing studies using a synthetic Whetstone benchmark," *Ada Lett.*, Vol. IV, No. 2, pp. 23–34, Sept. 1984.

[14] R. P. Weicker, "Dhrystone: A synthetic systems programming benchmark," *Commun. ACM*, Vol. 27, No. 10, 1984.

[15] A. R. Weiss, "Dhyrstone benchmark: History, analysis, scores and recommendations," ECL, LLC, Tech. Rep., 2002, Last accessed 6/2015. [Online]. Available: http://www.eembc.org/techlit/datasheets/dhrystone_wp.pdf

[16] "SPEC: Standard Performance Evaluation Corporation," Last accessed on 12/22/2014. [Online]. Available: www.spec.org

[17] "SPEC defect notice," Last accessed on 12/22/2014. [Online]. Available: http://www.spec.org/jbb2013/defectnotice.html

[18] "TOP500 supercomputer sites," Last accessed on 6/2015. [Online]. Available: http://www.top500.org

[19] "EEMBC: The Embedded Microprocessor Benchmark Consortium," Last accessed 6/2015. [Online]. Available: www.eembc.org

[20] S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *IEEE International Symposium on Workload Characterization (IISWC)*, 2009, pp. 44–54.

[21] H. Jin, M. Frumkin, and J. Yan, "The OpenMP implementation of NAS parallel benchmarks and its performance," October 1999, Last accessed 6/2015. [Online]. Available: https://www.nas.nasa.gov/assets/pdf/techreports/1999/nas-99-011.pdf

[22] M. A. Heroux, D. W. Doerfler, P. S. Crozier, J. M. Willenbring, H. C. Edwards, A. Williams, M. Rajan, E. R. Keiter, H. K. Thornquist, and R. W. Numrich, "Improving performance via mini-applications," Sandia National Laboratories, Tech. Rep. SAND2009-5574, 2009, Last accessed 6/2015. [Online]. Available: https://mantevo.org/MantevoOverview.pdf

[23] K. Barker, T. Benson, D. Campbell, D. Ediger, R. Gioiosa, A. Hoisie, D. Kerbyson, J. Manzano, A. Marquez, L. Song, N. Tallent, and A. Tumeo, *PERFECT (Power Efficiency Revolution For Embedded Computing Technologies) Benchmark Suite Manual*, Pacific Northwest National Laboratory and Georgia Tech Research Institute, December 2013, http://hpc.pnnl.gov/projects/PERFECT/, last accessed 6/2015.

[24] "Energybench version 1.0 power/energy benchmarks," Last accessed 6/2015. [Online]. Available: http://www.eembc.org/benchmark/power_sl.php

[25] F. Brglez and D. Fujiwara, "A neutral netlist of 10 combinational benchmark circuits and a target translator in FORTRAN," in *International Symposium on Circuits and Systems*, June 1985, pp. 695–698.

[26] D. Bryan, "The ISCAS '85 benchmark circuits and netlist format," MCNC, Tech. Rep., 1985.

[27] M. C. Hansen, H. Yalcin, and J. P. Hayes, "Unveiling the ISCAS-85 benchmarks: a case study in reverse engineering," *IEEE Design Test of Computers*, Vol. 16, No. 3, pp. 72–80, 1999.

[28] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *International Symposium on Circuits and Systems*, May 1989, pp. 1929–1934 vol.3.

[29] F. Corno, M. S. Reorda, and G. Squillero, "RT-level ITC'99 benchmarks and first ATPG results," *IEEE Design & Test of Computers*, Vol. 17, No. 3, pp. 44–53, 2000.

[30] M. S. Reorda, "ITC'99 benchmarks (2nd release)," Last accessed 6/2015. [Online]. Available: http://www.cad.polito.it/downloads/tools/itc99.html

[31] "IWLS 2005 benchmarks," Last accessed 6/2015. [Online]. Available: http://iwls.org/iwls2005/benchmarks.html

[32] 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.

[33] J. Sloan, R. Kumar, and G. Bronevetsky, "Algorithmic approaches to low overhead fault detection for sparse linear algebra," in *the proceedings of the 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, June 2012, pp. 1–12.

[34] P. Du, A. Bouteiller, G. Bosilca, T. Herault, and J. Dongarra, "Algorithm-based fault tolerance for dense matrix factorizations," *SIGPLAN Not.*, Vol. 47, No. 8, pp. 225–234, Feb. 2012. [Online]. Available: http://doi.acm.org/10.1145/2370036.2145845

[35] P. Rech, T. Fairbanks, H. Quinn, and L. Carro, "Threads distribution effects on graphics processing units neutron sensitivity," *IEEE Trans. Nucl. Sci.*, Vol. 60, No. 6, pp. 4220–4225, Dec 2013.

[36] D. Oliveira, L. Pilla, T. Santini, and P. Rech, "Evaluation and mitigation of radiation-induced soft errors in graphics processing units," *IEEE Transactions on Computers*, Vol. PP, No. 1, pp. 1–14, 2015.

[37] H. Quinn, T. Fairbanks, J. L. Tripp, and A. Manuzzato, "The reliability of software algorithms and software-based mitigation techniques in digital signal processors," in *IEEE Radiation Effects Data Workshop (REDW)*, 2013, pp. 1–8.

[38] J. Krüger and R. Westermann, "Linear algebra operators for GPU implementation of numerical algorithms," in *ACM Transactions on Graphics (TOG)*, Vol. 22, No. 3, 2003, pp. 908–916.

[39] J. Liepe, C. Barnes, E. Cule, K. Erguler, P. Kirk, T. Toni, and M. P. Stumpf, "ABC-SysBio–approximate Bayesian computation in Python with GPU support," *Bioinformatics*, Vol. 26, No. 14, pp. 1797–1799, 2010.

[40] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "GPU computing," *Proceedings of the IEEE*, Vol. 96, No. 5, pp. 879–899, 2008.

[41] S. S. Mukherjee, J. Emer, and S. K. Reinhardt, "The soft error problem: An architectural perspective," in *11th International Symposium on High-Performance Computer Architecture (HPCA-11)*, 2005, pp. 243–247.

[42] M. Desogus, L. Sterpone, and D. M. Codinachs, "Validation of a tool for estimating the effects of soft-errors on modern SRAM-based FPGAs," in *IEEE 20th International On-Line Testing Symposium (IOLTS)*, 2014, pp. 111–115.

[43] M. Aguirre, J. N. Tombs, A. Torralba, and L. G. Franquelo, "UNSHADES-1: An advanced tool for in-system run-time hardware debugging," in *International Conference on Field-programmable Logic and Applications (FPL)*, Vol. 2778, 2003, pp. 1170–1173.

[44] M. Aguirre, J. Tombs, F. Muoz, V. Baena, H. Guzman, J. Napoles, A. Torralba, A. Fernandez-Leon, F. Tortosa-Lopez, and D. Merodio, "Selective protection analysis using a SEU emulator: Testing protocol and case study over the Leon2 processor," *IEEE Trans. Nucl. Sci.*, Vol. 54, No. 4, pp. 951–956, 2007.

[45] H. Guzman-Miranda, M. Aguirre, and J. Tombs, "Noninvasive fault classification, robustness and recovery time measurement in microprocessor-type architectures subjected to radiation-induced errors," *IEEE Trans. Instrum. Meas.*, Vol. 58, No. 5, pp. 1514–1524, 2009.