A Practical Characterization of a NASA SpaceCube Application through Fault Emulation and Laser Testing

John Paul Walters, Kenneth M. Zick, and Matthew French
Information Sciences Institute, University of Southern California
Arlington, VA 22203
Email: {jwalters, kzick, mfrench}@isi.edu

***Abstract*—Historically, space-based processing systems have lagged behind their terrestrial counterparts by several processor generations due, in part, to the cost and complexity of implementing radiation-hardened processor designs. Efforts such as NASA's SpaceCube seek to change this paradigm, using higher performance commercial hardware wherever possible. This has the potential to revolutionize onboard data processing, but it cannot happen unless the soft error reliability can be characterized and deemed sufficient.**

A variety of fault injection techniques are used to evaluate system reliability, most commonly fault emulation, fault simulation, laser testing, and particle beam testing. Combining multiple techniques is more complex and less common. In this study we characterize a real-world application that leverages a radiation-hardening by software (RHBSW) solution for the SpaceCube platform, using two fault injection strategies: laser testing and fault emulation. We describe several valuable lessons learned, and show how both validation techniques can be combined to greater effect.

I. Introduction

The space community sits at an inflection point. As Earth-orbiting satellite imaging systems continue to generate massive volumes of data, either the satellite to Earth downlink capacity must increase to accommodate this data, or onboard processing must be used to reduce the data volume. Early experiments from NASA's EO-1 mission demonstrated the viability of onboard processing, and future missions expect to use it more extensively [1].

Typically, space-based computing leverages radiation-hardened processing elements (e.g. FPGAs, processors, memories). Processors such as the Mongoose V [2], the Rad6000 [3], and the Rad750 [4] provide this radiation -hardening at the cost of processing performance. To realize the full benefit of onboard processing, however, future missions will require much greater computational power than is available in today's radiation-hardened processors.

NASA has provided an alternative processing platform through the SpaceCube [1]. Instead of performing all data processing in radiation-hardened components, the SpaceCube can provide commercial systems-on-a-chip paired with a low performance radiation-hardened microcontroller. This allows scientific missions to utilize far greater processing power, provided that the SpaceCube-based processing reliability can be characterized and well understood, and provided that software approaches to fault tolerance can be proven effective.

Through our previous work we have developed an application-agnostic fault tolerance suite for embedded platforms like the NASA SpaceCube [5], allowing such platforms to mitigate control errors (detecting and correcting data errors is an active area of our research).

The focus of this paper is to characterize, through fault injection, the reliability of a real application with our radiation-hardening by software (RHBSW) solution targeting the SpaceCube platform. There are four common fault injection methods to characterizing a processor or system's reliability: fault emulation, fault simulation, laser testing, and particle beam testing. Particle beam testing is known to be the "gold standard" of radiation characterization for devices. In our case, however, particle beam testing was not suitable because it would have exposed portions of the chip that were not protected by our radiation-hardening by software scheme.

Fault simulation is commonly used by logic designers to characterize their own hardware designs, whether for space or terrestrial applications. This is convenient because such designers have access to RTL and netlists that enable accurate simulation. We do not have access to such proprietary information for the embedded CPU cores on the SpaceCube. Thus fault simulation was infeasible.

Instead, we base our characterization on a combination of laser testing and fault emulation. Laser testing allows for precise control of the injection target to the micron level, allowing injection of a single pulse at a specified $x-y$ coordinate. Fault emulation (injecting faults into an actual machine such as a prototype) provides an inexpensive solution to long running fault injection campaigns, enabling users to collect thousands of injections continuously over a period of days or weeks. We show how we can leverage both techniques to greater effect than either laser testing or fault emulation can provide alone.

As such, we make two contributions in this paper: 1) we offer lessons learned while characterizing the reliability of an embedded application using both laser testing and fault emulation, 2) we describe the synergistic effects achieved by combining laser testing and fault emulation.

II. Background

A. NASA SpaceCube and Onboard Data Processing

The NASA SpaceCube is a family of platforms created at the NASA Goddard Space Flight Center for high-performance onboard data processing [1]. By leveraging leading-edge commercial devices, FPGA-based acceleration, and radiation-hardening by software, this paradigm aims for breakthrough performance for key applications. A SpaceCube 1.0 platform

978-1-4799-0181-4/13/$31.00 ©2013 IEEE

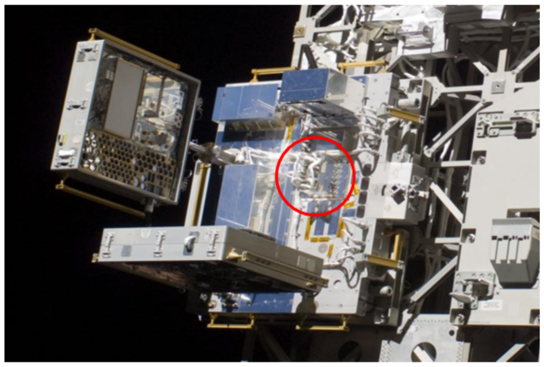


Fig. 1: SpaceCube (circled) on the International Space Station MISSE-7 experiment.

using four Xilinx Virtex-4FX60 FPGAs was installed on the International Space Station in 2009, as shown in Fig. 1. Another system flew on the Hubble Space Telescope Servicing Mission 4 in 2009. Next generation versions are being developed for a range of missions including CubeSats.

B. Error Detection and Recovery Methods

Our fault tolerance platform is composed of two primary error detection mechanisms: application-level control flow assertions, and heartbeat monitors. We assume the presence of a radiation-hardened controller like the one present on the SpaceCube 1.0 platform. The radiation-hardened controller performs no data processing; however, it is responsible for monitoring the heartbeats sent from each of the embedded PowerPC processors.

The primary recovery mechanisms are through our embedded checkpoint/rollback library. Recovery is performed either via self-recovery, where an individual PowerPC application recovers itself, or via the rad-hard microcontroller.

C. Fault Emulation

A fault emulator can be used to emulate upsets within the software-writable regions of a prototype system or processor. Unlike laser or particle beam-induced faults, emulated faults can be precisely specified through the emulator's logical view of the hardware with precise reporting of specific upsets and their logical location. This level of detail can prove incredibly valuable when characterizing an application or system, allowing developers to tune their fault tolerance strategies to the unique characteristics of a specific application. Another benefit of fault emulation is its low cost, enabling inexpensive long-running fault injection campaigns over days, weeks, or longer.

However, fault emulation has its limitations. The fault mechanism does not always closely mimic the fault mechanism of real world radiation-induced faults. Multi-bit upsets cannot typically be modeled accurately. Moreover, because this type of fault emulation can only characterize systems based on their software-writable regions, any non-software-visible regions are ignored. As we later show, this can lead to overestimating the impact of specific architectural features (e.g. the register set).

D. Laser Testing

A laser can be used to inject transient faults into an integrated circuit [6], enabling an evaluation of a system's fault tolerance methods. Unlike particle beam-induced faults, laser-induced faults can be finely controlled in space and time; single laser pulses can be injected into a system in order to test single event effects, and a laser can be readily focused on a circuit of interest with a spot size on the order of a micron. A laser pulse ionizes a circuit region, potentially causing states to flip in one or more sequential elements or combinational circuit nodes. The physical fault mechanisms are not identical to those involving particle collisions, so laser testing does not directly characterize the raw fault rates that can be expected in a particle radiation environment. However, the precise generation of transient faults allows for characterization of fault-to-error probabilities, and the scheduling of laser facilities is often easier and more flexible than scheduling of particle beam facilities.

Laser testing has advantages over fault emulation as well. Faults can be injected across all sequential elements as opposed to only those that are software-writable. Faults in combinational logic (single event transients aka SETs) can be injected. Importantly, it is also possible to generate realistic multi-bit upsets; this is usually not feasible with fault emulation when a system contains proprietary IP cores for which the logical-physical mapping is unknown.

III. Application, System, and Fault Models

A. System and Application Models

For testing, we modeled the SpaceCube using commercial Xilinx Virtex-4FX60 FPGAs identical to those on the SpaceCube 1.0 platform. Laser testing and fault emulation campaigns were conducted on separate test boards due to the requirements of the laser test. More details on the laser testing apparatus are described in Section IV. In both the laser and fault emulation campaigns, two PowerPC processors are used: one for application execution, and the other to model the radiation-hardened microcontroller.

Our application model synthesizes the major components of common satellite imagery applications: FFT, complex multiplication, and thresholding. We created an application representative of synthetic aperture radar (SAR) as well as hyperspectral image classification. The application repeatedly performs 1-dimensional FFTs and complex-complex multiplication followed by thresholding (see Fig. 2). The number of loop iterations was fixed at 330 in order to limit the application's runtime to approximately 5 seconds.

At system startup a golden output is calculated that is used to verify results after both the fault emulation and laser testing trials. We define a trial as a single execution of our application, during which a single fault is injected. In the case of our laser test, a single pulse is emitted causing a single laser strike, while in the case of our fault emulator, a single state bit is flipped at random.

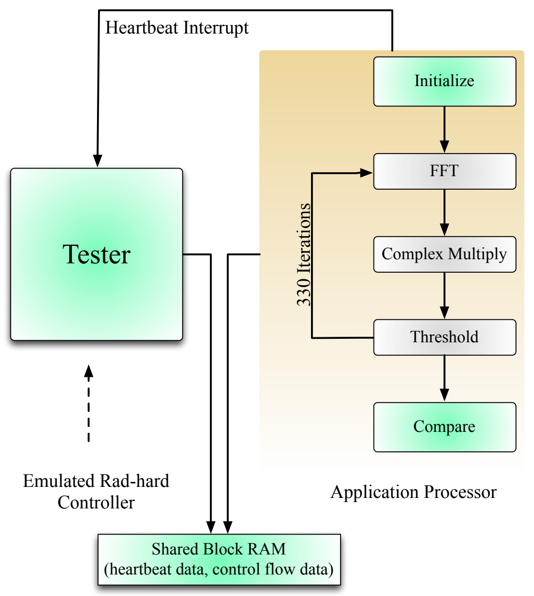


Fig. 2: Application model and its interaction with the emulated rad-hard controller.

At the end of each trial, the application PowerPC is reset, and the execution behavior is classified as one of the following error types:

- *Benign*: The application terminates normally within some threshold of the execution time. The results are correct.
- *Unrecoverable crash/hang*: The application crashes or hangs and cannot be reset either by the application PowerPC or the emulated rad-hard microcontroller. In this case, the test FPGA must be reprogrammed.
- *Recoverable crash/hang*: The application crashes or hangs, but is able to recover either through a rollback or a reset. If the rad-hard microcontroller intervenes, we consider this a recoverable crash/hang.
- *Silent data corruption (SDC)*: The application terminates normally, but the results are incorrect. This is not a specific focus of this work, however we track these cases for future investigation.

B. *Fault Models*

In our fault emulator, faults are modeled as an SEU (single event upset). Faults can be injected into the instruction and data caches, including both tags and cache line data as well as the register sets. Cache upsets are especially critical because Virtex-4 caches have non-functional cache parity circuits [7], a design fault of the embedded processor core. The fault is injected through an interrupt in a non-cacheable memory region, meaning the application is effectively paused while a memory element is selected at random and the state flipped.

For laser testing, faults are modeled as an SEU or SET at a random location within a region containing the embedded processor core. Only a single laser pulse is injected during one execution of the application of interest, with timing that is uniformly random across the execution time. Unlike in fault emulation, laser-induced SEUs may be multi-bit. Specific details about the laser testing methodology are provided in the following sections.

IV. Experimental Methodology

A. *Fault Emulation*

Fault emulation is performed using a custom designed PowerPC fault emulator. The experiment runs on a single Virtex-4FX60 FPGA. Testing is performed using a Xilinx ML410 development board. As described in Section III we use a single PowerPC core within the FPGA as the application processor, and a second core as an emulated rad-hard microcontroller. Faults are injected using a non-cacheable interrupt with non-cacheable variables to avoid polluting the application cache contents. Register upsets are implemented through a combination of inline assembly code and stack manipulation to inject upsets into the nonvolatile registers of the processor.

To inject upsets into the cache contents, a custom FPGA circuit is used. First, the target cache element and cache line are identified. In the case of the instruction cache, the line is invalidated and retouched. In doing so, the custom FPGA circuit modifies the bus transaction to insert the bit flip before the data enters the processor. A similar process is used in the case of the data cache; however, additional work is necessary to account for dirty cache lines. We provide a more detailed description of the fault emulator in our prior work [8].

The fault emulator is capable of injecting faults into any software-writable memory location. Fault locations are chosen by randomly selecting a bit among all software-writable bits. In Table I we provide an estimation of all of the bits within the PowerPC 405. We classify them by those accessible by our fault injector and those that are not.

TABLE I: PowerPC 405 sensitive bits

Feature	Size Accessible	Size Inaccessible
Instruction Cache	16KB + 1472B tag/ctrl	512 ctrl bits
Data Cache	16KB + 1344B tag/ctrl	512 ctrl bits
Registers	75 x 32 bits	0
Execution Pipeline	0	10 x 32 bits
ALU/MAC	0	~1200 bits
Timers	0	3 x 64 bits
MMU	0	72 x 68 bits
Misc	0	1024 bits
Total	**287,072 bits**	**8,656 bits**

From Table I we can see that most of the PowerPC 405's known sensitive bits are accessible to our fault emulator. Moreover, the vast majority of them exist within the PowerPC's instruction and data caches, making fault emulation within the caches especially critical. Still, our fault emulator is unable to access any of the ALU/MAC bits, execution pipeline bits, etc. This, broadly, is perhaps the primary disadvantage of fault emulation: critical parts of the PowerPC are not visible to the fault emulator.

The primary advantages of fault emulation, however, are the detailed feedback that is available after each emulated SEU, and the volume of trials that may be captured. Both the emulated radiation-hardened microcontroller and application send output via their own UART. After each trial, we know when the fault was injected, what memory region that fault was injected into, and the specific bit that was flipped. We ran three fault emulation campaigns and collected over ten thousand injections in total. We describe the campaigns and results in detail in Section V.

B. Laser Testing

An FPGA-based hardware/software tester was created largely from scratch to control the experiments. The tester includes a re-purposed high-speed digital tester board (originally created by NASA and ISI for particle beam testing [9]), a Xilinx Virtex-II Pro FPGA design, a test program that runs on a PowerPC core embedded in the Virtex-II Pro, and a high-density connector for interfacing to the system under test. The application under test runs on a PowerPC 405 core embedded in a Virtex-4FX60 FPGA daughtercard positioned in the path of the laser.

Two commercial Xilinx Virtex-4FX60 devices were prepared for laser testing by a third party (FA Instruments, Inc.). The devices were de-lidded and the backsides of the die were polished to mirror surface quality.

The laser testing was performed at the U.S. Naval Research Laboratory using a titanium sapphire pulsed laser (Clark-MXR CPA1000) in a dual-photon setup. The laser generated 150 fs pulses at a rate of 1 kHz with a spot size of 1.35 μm. The NRL laser had been calibrated in the week prior to testing to an intensity level of 73.6 pJ/mV, when using a neutral density filter (OD1) that reduces the intensity by 8.64$\times$ at the wavelength of interest.

Laser pulses were injected through the backside of the device, as depicted in Fig. 3. A view of the laser testing setup is provided in Fig. 4, with a close-up of the lens and device in Fig. 5.

One of the challenges with application-level testing is to inject only a single fault per application execution (e.g. once per 5 seconds). A laser pulse was generated every ms, so a means of selecting individual pulses was needed. NRL did not previously have this capability, so a solution had to be engineered. The same problem has been addressed at other facilities [10]. In our solution, the tester drives a trigger signal to a shutter controller which in turn opens and closes the shutter in the laser path. The shutter (Uniblitz LS2) only remains fully open for 0.7 ms and requires 1.5 ms to open and close. The tester carefully synchronizes the trigger timing to the laser to prevent a pulse from getting partially blocked and to prevent two partial laser pulses (spaced out by 1 ms) from getting through. The amount of laser energy delivered through the shutter was measured using a photodetector as quite consistent, confirming that individual pulses were cleanly selected.

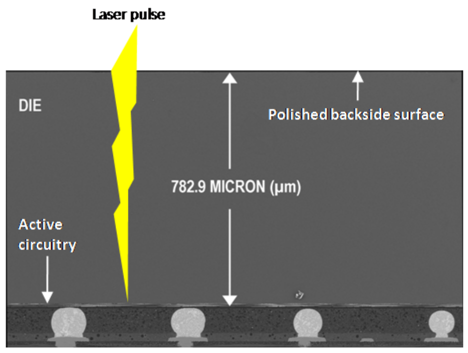
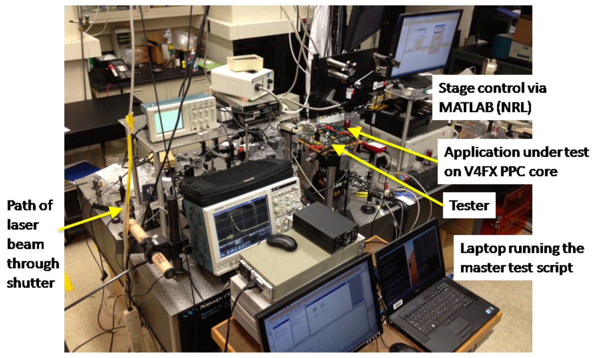


Fig. 3: Cross-section of a Virtex-4FX60 device depicting how a laser pulse is injected into the circuitry.


Stage control via MATLAB (NRL)
Application under test on V4FX PPC core
Tester
Laptop running the master test script
Path of laser beam through shutter
Fig. 4: Picture of laser testing setup.

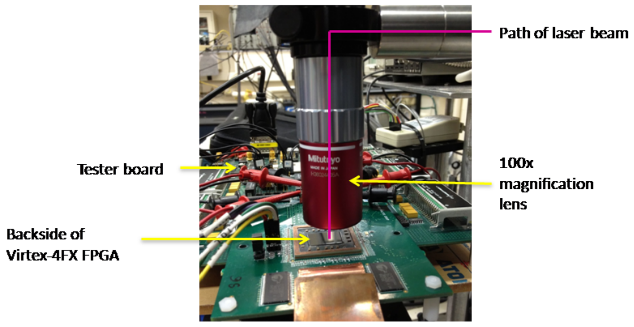


Fig. 5: Close-up picture of laser testing setup.

Initially, static testing was used to locate the instruction and data cache arrays within the PowerPC core area, to aid in tuning the laser energy, and to determine the number of bit flips caused by each injected laser pulse. The instruction and data caches were disabled, loaded with known patterns, and left in that state indefinitely as laser pulses were injected by manually triggering the shutter. Using the Xilinx Microprocessor Debugger (xmd) and debug instructions, the state of the caches was continuously monitored for bit flips, and any miscompares logged.

The main testing performed was dynamic testing of the application. At the beginning of a trial, the hardware was automatically positioned by the NRL's MATLAB program such that the laser beam path was pointed at a random location within the PowerPC core area. The tester reset the application, allowing it to begin executing. At a random time during the expected execution time window, the tester allowed a single laser pulse to be injected. The tester monitored the application and logged the resulting behavior, along with the laser pulse position, timing, and measured energy. Then the entire process was repeated. Each trial lasted 5 seconds plus any additional recovery period (e.g. to allow for rolling back and restarting from a checkpoint), corresponding to a few hundred trials per hour.

V. Experimental Results

In this section we describe four experiments. In the first experiment, a static laser test was used for calibrating our laser testing and validating the occurrence of bit upsets. The second experiment, a dynamic laser test, represents our primary laser test, where we validate our fault tolerance library against randomly injected laser pulses. The third experiment was a new fault emulation campaign, and the fourth combines elements of our dynamic laser test and fault emulation to validate the realism of our fault emulation experiments.

A. Static laser test

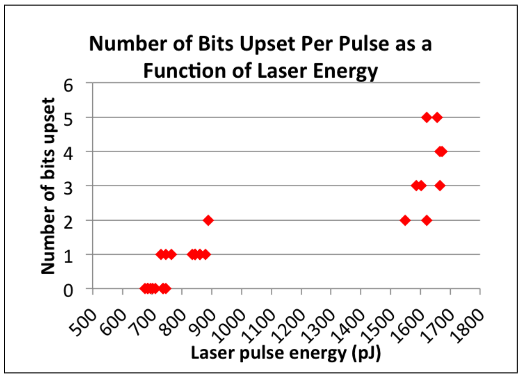


Fig. 6: Number of upsets observed for increasing laser intensities.

Having identified the instruction and data cache arrays (see Section IV) we began by injecting single laser pulses of increasing energy into a single location in the instruction cache array. In Fig. 6 we provide results from our static test. At low laser energies ($<$ 700 pJ), no upsets were detected. At 809 pJ, pulses would regularly cause a single bit to flip. At higher energies such as 1619 pJ, each pulse caused multiple bit flips (2-5). Note that the gap in Fig. 6 represents laser energies that were skipped over and does not imply that no upsets occurred at these energies. A full characterization of the cache was beyond the scope of this study.

Using this data, we settled on the 1619 pJ laser energy for our dynamic tests. This provided us with reasonable confidence that each laser pulse was causing an upset within the PowerPC core. This is critical because upsets are not directly observable during dynamic testing (only indirectly through application-level error behavior), and as we show in the following subsection, the vast majority of upsets caused no application-level misbehavior.

B. Dynamic laser test

The dynamic test represents our primary data from the laser experiment. Each trial is categorized as described in Section III. To maintain laser focus and alignment, the scan region of a single campaign was limited to just half of the PowerPC core area; two campaigns (left and right) were needed to scan the entire area.

TABLE II: Laser left and right half results

Error Class	Left	Right	% of Total	Total
Unrecoverable crash/hang	**0**	**0**	**0%**	**0**
Recoverable crash/hang	5	48	2.63%	53
Silent data corruption	20	27	2.33%	47
Benign	985	934	95.05%	1919
Total				2019

In Table II we provide the categorical output for each laser pulse injection in both the left and right side campaigns. Of particular note is that zero unrecoverable crashes/hangs were encountered; upon a crash/hang, the application was always successfully restarted from a checkpoint or through a reset.

In Fig. 7 we show each fault overlaid on an infrared die photo of the embedded PowerPC core. The non-benign upsets fall almost entirely within the PowerPC caches; so much so, that we can clearly identify the caches on the right side of the core. These are the instruction caches. The data caches, on the left side, are less obvious, but left side errors still fall almost exclusively within the data cache arrays. The remainder of the core is composed of the ALUs, MMUs, an Ethernet controller, register sets, etc. Notably, very few errors occurred in these regions. Our application uses neither the MMU nor Ethernet controller.

C. Fault emulation

TABLE III: Fault emulation full campaign results

Error Class	% of Total	Total
Unrecoverable crash/hang	0%	0
Recoverable crash/hang	13.27%	449
Silent data corruption	16.58%	561
Benign	71.15%	2374
Total		3384

Our third and fourth experiments center around the use of a software-based fault emulator to characterize the same application without the use of the laser. Our overall results, including injections into both caches and the register sets are shown in Table III. At a glance, it would appear that our fault emulator tends to overestimate the number of predicted application-level errors, suggesting that recoverable errors and silent data corruption should account for 13.3% and 16.6% of the errors, respectively. We explore this in more detail in the following section; however, it is worth noting that this demonstrates one of the major shortcomings of fault emulation:

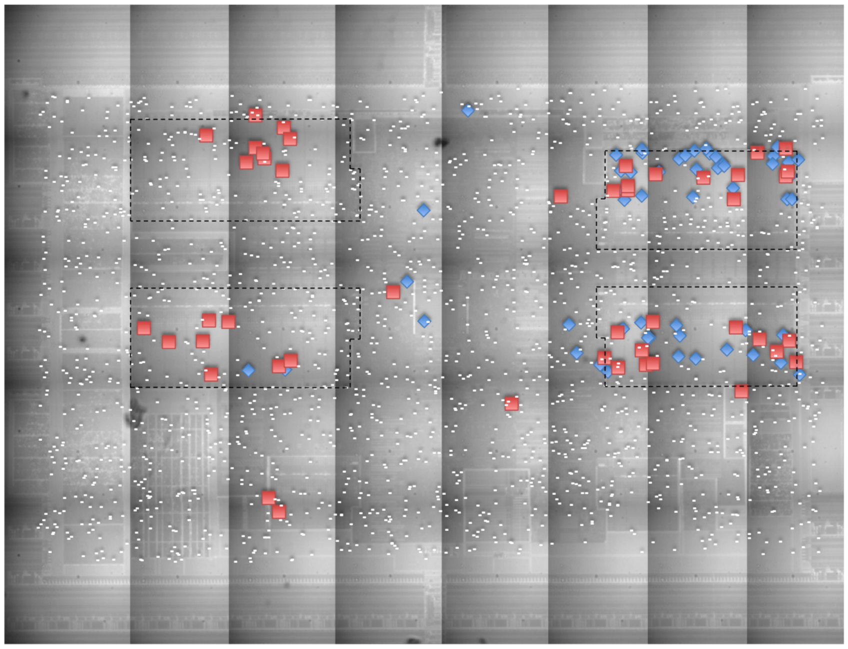


Fig. 7: Composite die photo showing the locations of benign faults (dots) and system-level errors overlaid on the embedded processor core. Caches are traced with dashed lines. The red squares represent silent data corruption and blue diamonds represent recoverable errors

the implied assumption that the software-writable memory regions represent the entirety of the vulnerable portions of the processor.

Further, close examination of Fig. 7 shows that the cache arrays actually account for much less of the processor area than a simple accounting of vulnerable bits would suggest (Table I). Indeed the cache arrays occupy less than half of the total area of the PowerPC core, suggesting that applying a derating factor to the probability of selecting a cache upset would improve the accuracy of the software-based fault emulator.

D. *Correlation between laser testing and fault emulation*

Because nearly every non-benign error occurred within one of the cache arrays, our final experiment focuses on validating our fault emulator against strictly the data and instruction caches. In this experiment, we ran two fault emulation campaigns, one against the data cache and another against the instruction cache. Under these experiments, we avoid the issue of cache area relative to the PowerPC core by injecting faults in *only* the instruction cache or data cache, and comparing those results against those laser faults that fell within the cache areas.

We ran 3956 and 5719 trials for the dedicated Icache and Dcache fault emulation campaigns, respectively. In Figs. 8 and 9 we compare our cache-specific fault emulator campaigns to our laser results. By comparing only the percent of errors, we find that the fault emulation results compare most favorably with our laser results in the case of recoverable errors.

However, because we are able to produce far more emulated injections than occurred within the cache arrays during our laser experiment, we also compare the results by averaging

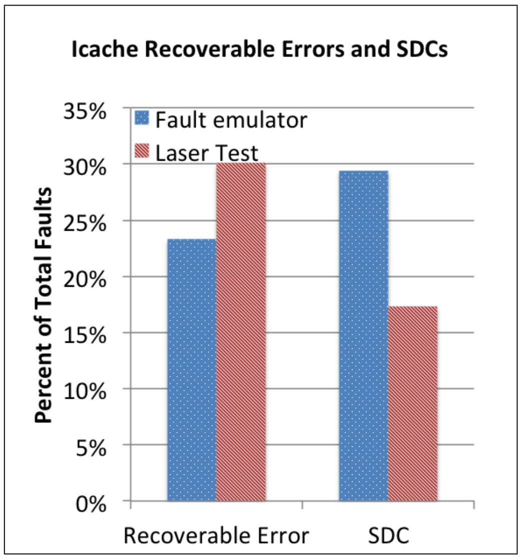


Fig. 8: Results of a dedicated Icache campaign compared to laser results. Values are listed as the percent of total faults.

blocks of comparably sized emulated injections to their laser counterparts. For example, there were 133 laser-based faults injected into the instruction cache and 92 laser-based faults injected into the data cache.

Using this analysis we can directly compare the emulated results with our laser experiment results. All results are within a 90% confidence interval. For our data cache results, recover-

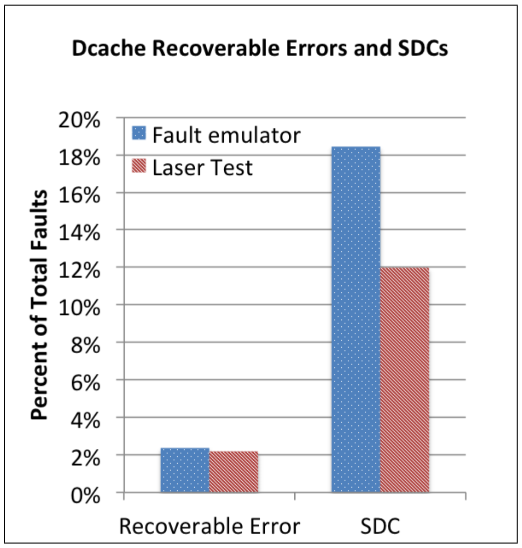


Fig. 9: Results of a dedicated Dcache campaign compared to laser results. Values are listed as the percent of total faults.

able errors are within 0.106σ (standard deviations), while the SDCs are within 1.367σ. For our instruction cache campaign, recoverable errors are within 1.640σ while our SDCs are within 1.371σ. This helps to confirm our cache injection model, and demonstrates the viability of fault emulators for characterizing SEUs.

VI. Lessons Learned

One of the primary goals of this paper is to characterize a single application through two fault injection methodologies and show how their combination can be leveraged to greater effect. Through both laser testing and fault emulation, we have shown the relative advantages and disadvantages of both techniques. In this section, we discuss the broad lessons that we have learned through each technique individually, and most importantly through the combination of both strategies.

A. Laser lessons

One important lesson learned is the amount of preparation and development required for system-level laser testing. Although we re-used a tester board previously designed for particle beam testing, the new FPGA logic, the embedded test program, and the capability for selecting individual laser pulses with a shutter all required significant effort. We initially found it very difficult to get FPGA devices polished; ultimately a third party was able to provide prompt service.

An unexpected lesson that we learned early on was the importance of maintaining laser alignment throughout the entirety of the experiment. When out of alignment, upsets were not reliably generated. Thus the alignment needed to be checked between each campaign, and the size of the scan region had to be limited such that the device did not move unintentionally in the z direction. This was the reason that we split our injection campaigns into two regions.

B. Fault Emulation Lessons

The primary lesson learned from our fault emulation experiments was that future studies of this kind must incorporate laser testing and/or particle beam testing into their fault distribution models in order to more accurately model the different error rates in SRAM cells, registers, etc. While an error distribution based on Table I represents a starting point, fully characterizing error rates will require more detailed fault models than are currently available.

As a practical matter, fault emulation proved valuable as a precursor to our laser test. The fault distribution was useful for studying application behavior in the presence of bit errors. By using fault emulation, we were able to implement a robust fault tolerance solution prior to engaging the NRL for laser testing. By doing so, we were able to test our solution without debugging the fault tolerance library in-place. This resulted in far more productive testing and many more trials over the three day testing period.

C. Combining Laser Testing and Fault Emulation

While both fault emulation and laser testing, individually, proved useful, neither provided a full characterization of our SpaceCube-based application. Broadly, software-controlled fault emulation tended to overemphasize cache and register errors, while laser testing provided only a small sample of faults.

Characterizing an application as complex as a SpaceCube-based embedded processor application requires thousands more trials than can reasonably be collected through laser testing. However, laser testing provides a realistic fault model that cannot be replicated through fault emulation without access to proprietary design details.

The combination of fault emulation and laser testing has allowed us to more fully explore the failure space of the system under test. Fault emulation allows us to gather tens of thousands of trials, over weeks if desired, and at extremely low cost. Our fault emulation campaigns also correctly predicted that for the application under test, the instruction cache is far more sensitive than the data cache in terms of recoverable errors. The cache injection campaigns proved accurate when compared to their laser counterparts. This was especially true in the case of recoverable errors (see Figs. 8 and 9). This was true despite the somewhat unrealistic fault model imposed by the fault emulator, the inability to accurately model multi-bit upsets in emulation, and the fact that each injection occurs along clean clock boundaries, essentially while the system is paused (in an interrupt).

Laser testing provides a far more realistic fault mechanism, injecting faults independent of processor state, causing multi-bit upsets, and injecting into logic that is unavailable to a software-based fault emulator. By combining these two techniques, we were able to leverage the benefits of both techniques to explore a much greater portion of the failure space than either is capable of achieving alone.

Finally one insight gained from both laser testing and fault emulation was the relative effectiveness of error detection on both the application processor and the emulated radiation-hardened controller. We found that a surprising majority of detected recoverable errors were caught by the application processor: 94% in the case of our laser experiment, and 97.5% in the case of our full campaign fault emulation experiment (Table III). The emulated radiation-hardened controller was responsible for detecting 6% in the case of the laser experiment and 2.5% in the case of the fault emulation campaign.

VII. Related Work

We are not aware of studies that specifically address the correlation between fault emulation and system-level laser testing, however many previous studies have related aspects. Evaluations of software-implemented fault tolerance in embedded applications have been conducted with in situ data [11]. Several studies correlate fault simulation and particle beam testing [12], or fault emulation and particle beam testing [13]. In some cases, lasers have been used to perform dynamic, application-level testing, often in the context of cryptographic applications [10]. Methods of software-based fault injection have been well documented. Carreira et al. demonstrated a fault injector using a CPU's debug features and exceptions [14]. They were able to inject faults into registers and functional units (but not caches). Velazco et al. used interrupts to inject faults into registers and on-chip RAM [15]. A method for injecting faults into both registers and caches on a embedded PowerPC core was described by Bernardi et al. [16], and we have developed our own fault injector, used in this paper [8]. The use of checkpoint/restart in embedded systems has been previously described by Pop et al. [17]. In our earlier work we focused on fault tolerance [5] and silent data corruption [18] in a NASA SpaceCube application.

VIII. Conclusion

In this paper we have made two contributions: 1) we have shared the valuable lessons learned in characterizing an embedded application though both laser testing and fault emulation; and 2) we showed how combining both fault emulation and laser testing results in a greater understanding of the failure space than either technique alone provides. A secondary result of this paper is that we have demonstrated the robustness of our fault tolerance solution, tolerating thousands of faults and recovering from every one. Our hope is that this study will offer insight to researchers who may want to use laser testing and/or fault emulation for their own work, and who wish to employ both techniques more effectively.

Acknowledgments

The authors would like to acknowledge the contributions of the following individuals and organizations for their assistance in preparing this manuscript and carrying out the laser experiments. Scott Stansberry of USC/ISI provided hardware guidance. Jonathan Ahlbin of USC/ISI provided valuable feedback on this manuscript. Stephen Buchner and Jeffrey Warner of the U.S. Naval Research Lab provided expertise in laser testing. Mark Bucciero designed and implemented much of the framework used in our fault emulation work. NASA provided a test board and DUT cards. Finally, the authors greatly acknowledge Tom Flatley, the NASA Goddard Spaceflight Center, and the NASA Earth Science Technology Office AIST Program for funding this work through grant number NNX09AF16G.

References

[1] T. Flatley, "Advanced hybrid on-board science data processor-SpaceCube 2.0," *Earth Science Technology Forum*, 2011.

[2] Synova Inc. (2012) Mongoose-V MIPS R3000 Rad-Hard Processor. [Online]. Available: http://www.synova.com/proc/mg5.html

[3] N. Haddad, R. Brown, T. Cronauer, and H. Phan, "Radiation hardened COTS-based 32-bit microprocessor," in *Fifth European Conference on Radiation and Its Effects on Components and Systems, RADECS 99*. IEEE, 1999, pp. 593–597.

[4] D. Rea, D. Bayles, P. Kapcio, S. Doyle, and D. Stanley, "PowerPC RAD750-A Microprocessor for Now and the Future," in *IEEE Aerospace Conference*. IEEE, 2005, pp. 1–5.

[5] M. Bucciero, J. P. Walters, and M. French, "Software fault tolerance methodology and testing for the embedded PowerPC," in *IEEE Aerospace Conference*, Mar. 2011.

[6] S. P. Buchner, D. Wilson, K. Kang, D. Gill, J. A. Mazer, W. D. Raburn, A. B. Campbell, and A. R. Knudson, "Laser Simulation of Single Event Upsets," *IEEE Transactions on Nuclear Science*, vol. 34, no. 6, pp. 1227 –1233, Dec. 1987.

[7] Xilinx, "Virtex-4 PowerPC 405 - Errata for all Virtex-4 FX Devices."

[8] M. Bucciero, J. P. Walters, R. Moussalli, S. Gao, and M. French, "The PowerPC 405 Memory Sentinel and Injection System," in *Proceedings of the 19th Annual International Symposium on Field-Programmable Custom Computing Machines*, ser. FCCM '11. IEEE Computer Society, 2011, pp. 154–161.

[9] C. Poivey, M. Berg, M. Friendlich, H. Kim, D. Petrick, S. Stansberry, K. LaBel, C. Seidleck, A. Phan, and T. Irwin, "Single Event Effects (SEE) response of embedded PowerPCs in a Xilinx Virtex-4 FPGA for a space application," in *9th European Conference on Radiation and Its Effects on Components and Systems, RADECS 2007*, Sep. 2007, pp. 1 –5.

[10] V. Pouget, A. Douin, G. Foucard, P. Peronnard, D. Lewis, P. Fouillat, and R. Velazco, "Dynamic Testing of an SRAM-Based FPGA by Time-Resolved Laser Fault Injection," in *14th IEEE International On-Line Testing Symposium IOLTS '08*, July 2008, pp. 295 –301.

[11] M. N. Lovellette *et al.*, "Strategies for fault-tolerant, space-based computing: Lessons learned from the ARGOS testbed," in *IEEE Aerospace Conference*, 2002.

[12] V. Asenek, C. Underwood, R. Velazco, S. Rezgui, M. Oldfield, P. Cheynet, and R. Ecoffet, "SEU induced errors observed in microprocessor systems," *IEEE Transactions on Nuclear Science*, vol. 45, no. 6, pp. 2876 –2883, Dec. 1998.

[13] P. Ramachandran, P. Kudva, J. Kellington, J. Schumann, and P. Sanda, "Statistical Fault Injection," in *International Conference on Dependable Systems and Networks, DSN 2008*, June 2008, pp. 122 –127.

[14] J. Carreira, H. Madeira, and J. Silva, "Xception: A Technique for the Experimental Evaluation of Dependability in Modern Computers," *IEEE Trans. Softw. Eng.*, vol. 24, no. 2, pp. 125–136, 1998.

[15] R. Velazco, S. Rezgui, and R. Ecoffet, "Predicting error rate for microprocessor-based digital architectures through C.E.U. (Code Emulating Upsets) injection," *IEEE Transactions on Nuclear Science*, vol. 47, no. 6, pp. 2405 –2411, Dec. 2000.

[16] P. Bernardi, L. Sterpone, M. Violante, and M. Portela-Garcia, "Hybrid Fault Detection Technique: A Case Study on Virtex-II Pro's PowerPC 405," *IEEE Transactions on Nuclear Science*, vol. 53, no. 6, pp. 3550 –3557, Dec. 2006.

[17] P. Pop, V. Izosimov, P. Eles, and Z. Peng, "Design Optimization of Time- and Cost-Constrained Fault-Tolerant Embedded Systems With Checkpointing and Replication," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 3, pp. 389 –402, Mar. 2009.

[18] K. Zick, C.-C. Yu, J. P. Walters, and M. French, "Silent Data Corruption and Embedded Processing With NASA's SpaceCube," *IEEE Embedded Systems Letters*, vol. 4, no. 2, pp. 33 –36, June 2012.