The PowerPC 405 Memory Sentinel and Injection System

Mark Bucciero, John Paul Walters, Roger Moussalli, Shanyuan Gao, Matthew French
University of Southern California
Information Sciences Institute
Arlington, VA USA
{*mbuccier, jwalters, rmoussalli, sgao, mfrench*}*@isi.edu*

***Abstract*—Traditional approaches to evaluating a system's vulnerability to Single Event Upsets (SEUs) require elaborate and costly radiation beam testing or time-consuming simulation. While beam testing represents definitive evidence of a processor's susceptibility to radiation-induced upsets, we believe that low-cost in-house bit error injection tests provide a valuable tool both in their own right and as an intermediate step towards approximating a processor or application's behavior in the presence of cosmic radiation, prior to beam testing.**

In this paper we describe a new hardware/software tool named the Memory Sentinel and Injection System (MSIS) that initially targets the two PowerPC 405s within the Xilinx Virtex-4 FX family of FPGAs. The MSIS leverages the configurable logic of an FPGA as well as a custom software interrupt to inject bit errors into the full set of a processor's registers and caches in a user-transparent fashion. By using the second PowerPC as a monitor, the MSIS is capable of performing rapid and unattended fault injection. Through 20,000 injections we demonstrate the impact of the MSIS on a sample sensor-like application and analyze its behavior in the presence of upsets.

***Keywords*-FPGA, SEU, Fault Injection, PowerPC**

I. Introduction

Recent generations of Xilinx FPGAs have moved to heterogeneous architectures that contain configurable logic, memories and FIFOs, multiply and accumulate units, ethernet cores, high-speed serial transceivers, and embedded PowerPC microprocessors. This architectural flexibility has allowed for the development of systems on a chip (SoCs) within an FPGA. Using the embedded processors as a part of the SoC has created an easy migration path for existing C programs to FPGA-based platforms. An FPGA-based SoC is an attractive option for the space community's scientific applications because the PowerPC in an FPGA can perform more computations per second than the leading radiation hardened microprocessor (see Table **??**). However, using an FPGA in a space-based system will expose it to radiation which can cause Single Event Upsets (SEUs). An SEU in an FPGA may cause one or more memory cells to change state. If this state change occurs within the configuration memory of an FPGA, a functional difference may be observed until the device is reprogrammed.

This work was supported by NASA grant #NNX09AF16G.

When an FPGA is used in a space-based system, a number of fault tolerance strategies [**?**], [**?**], [**?**], [**?**], can detect and correct errors within the FPGA's bitsteam. However, the PowerPC cores are uniquely vulnerable to SEUs because the bitstream does not provide access to the run-time state of the RISC core. As a result, traditional configurable logic fault injection tools are ineffective at testing the PowerPC [**?**], [**?**], [**?**], [**?**]. Being able to test for these processor upsets is critical for space applications, where SEUs are common and computational resources are limited. In this paper, we introduce the Memory Sentinel and Injection System (MSIS, pronounced em-sis) as a method for software-based fault injection to the PowerPC 405 within the Xilinx Virtex-4 FX series of FPGAs. The fault injection uses a bit flip to emulate an SEU within the PowerPC rather than the configurable logic in the FPGA. The software that injects the fault runs on the target processor, so it is important for the MSIS to maintain a level of application transparency during the injection routine.

The main contribution of this paper is a proto-type software-based fault injector that is capable of executing applications on hardware while injecting single bit errors into the register sets and caches during real time execution. We then describe a hardware-based monitor that may be used to mitigate faults that attempt to overwrite critical memory segments. Finally, we demonstrate the MSIS fault injector against a sample application running on a PowerPC 405 processor.

This paper is organized as follows. Section 2 describes some of the previous work related to both FPGA fault injection and software-based fault injection. Section 3 describes the PowerPC 405 that is contained with a Xilinx Virtex-4 FX FPGA and the bits that can be controlled through software. Section 4 introduces the MSIS architecture and explains the different methods for fault injection and memory protection. Then, Section 5 presents our fault injection experiment and its associated results. Finally, Section 6 draws some conclusions and looks to some future research.

II. Related Work

When operating in a space environment, Xilinx SRAM based FPGAs, like other SRAM memories, are susceptible to radiation induced Single Event Upsets (SEUs) [**?**], [**?**].

SRAM FPGAs are somewhat unique devices as SEUs can affect registers in either the configuration memory, which holds the state of the circuit, or the functional plane, which holds the users state elements. These resources are generally protected from SEUs by TMR and bitstream scrubbing. To test the configuration memory, an artificial upset can be injected by flipping bits in the configuration bitstream and observing the results [**?**]. Using this method, the reliability of a particular application can be measured in terms of configuration bits. The effectiveness of bitstream scrubbing and TMR of an application can also be determined by comparing the original implementation results to the fault protected implementation results.

Unlike the other computational elements in the FPGA, the PowerPC 405 is not fully observable from the configuration memory. Scrubbing and TMR can be used to protect the logic surrounding the PowerPC, but cannot be used for the PowerPC internal registers, providing a significant hurdle for space-based use. Limited research has been done in this area, with the Simple Portable Fault Injector for the Embedded PowerPC (SPFI-ePPC) [**?**] as a wrapper around GDB, to test how the PowerPC responds to a fault. SPFI-ePPC sets a breakpoint randomly during an application, changes a value, and resumes program execution. This method of fault injection is attractive for testing because it does not involve changing the HDL of a design. However, it can only modify registers and memory that are writable through GDB. The general and special purpose registers only account for a small percentage of the sensitive bits within the PowerPC, when the instruction and data caches are enabled (see Table **??**). SPFI-ePPC provides a good first level analysis of how an application will respond to a bit-flip.

A software fault injection method called Code Emulating an Upset (CEU) has been implemented on the 80C51 microprocessor and the 320C50 digital signal processor [**?**]. To inject a CEU, the following steps are performed on the processor:

1) Assert an interrupt to force an injection
2) Save the context of the processor
3) Perform a bit flip
4) Restore the application context

This technique has been found viable when comparing software injected faults to radiation induced SEUs and is similar to the MSIS injection method. However, the processors under test did not have cache memory.

III. PowerPC 405 Architecture

The current MSIS implementation targets Xilinx's proprietary PowerPC 405 architecture. While the majority of the MSIS software components may be generalized to the PowerPC family of microprocessors, several of our injection strategies (notably cache injections) are unique to a processor embedded within an FPGA. In this section we provide a brief overview of the PowerPC 405 architecture.

Table I
PERFORMANCE OF THE POWERPC 405 AND COMMON RADIATION HARDENED PROCESSORS

Processor	**MIPS**
RAD6000	35
RAD750	266
LEON3FT	560
PowerPC 405 (2)	900

Table II
POWERPC 405 SENSITIVE BITS

Feature	**Size**
Instruction Cache	16KB + 1408B tag + 64 control bits
Data Cache	16KB + 1216B tag + 64 control bits
General Purpose Register Set	32 x 32 bits
Special Purpose Register Set	32 x 32 bits
Execution Pipeline	10 x 32 bits
ALU/MAC	1200 bits
Timers	3 x 64 bits
MMU	72 x 68 bits
Misc	1024 bits
Total	**292,820 bits**

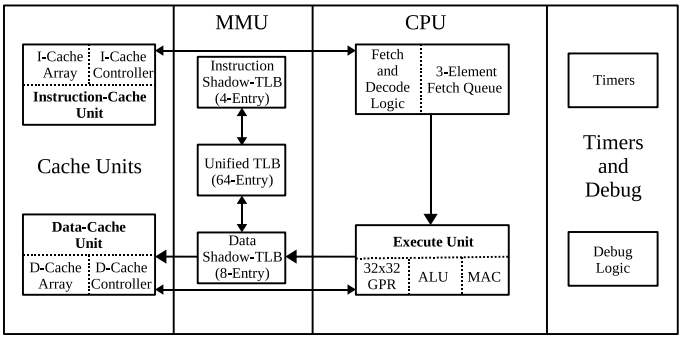

Figure 1. Xilinx PowerPC 405 block diagram, adapted from [**?**].

The Xilinx Virtex-II Pro and Virtex-4 devices contain two PowerPC 405 processors. Each PowerPC is a hard core, 32-bit RISC, Harvard architecture processor (see Figure **??**). The instruction and data caches are both 16KB, 2-way set associative. Each cache line is 256 bits wide. The caches may be configured in either write-through or write-back mode. The PowerPC 405 also includes a software controlled memory management unit (MMU); however, the MMU is typically only used by an operating system.

While not radiation hardened, the PowerPC 405 is an attractive space processor due its low cost and high performance. In Table **??** we list several common radiation hardened space processors along with their respective performance. Notably, the combined processing performance of the Virtex 4's PowerPC 405s outperforms all of the radiation hardened processors described in Table **??** by

a wide margin: 25x, 3.4x, and 1.6x for the RAD6000, RAD750, and LEON3FT, respectively. The challenge in using the PowerPC 405s, however, is that unlike the radiation hardened processors described in Table **??**, they are vulnerable to radiation effects.

To help characterize its sensitivity to radiation upsets, we developed an estimate of the PowerPC 405's sensitive bits, shown in Table **??**. While the precise details of Xilinx's design are proprietary (see [**?**]), these estimates are based on available documentation as well as our own experience implementing RISC processors [**?**], [**?**]. We caution that our analysis does not imply that any particular bit within Table **??** is more sensitive than any other. Rather, Table **??** is meant to reveal only the relative size of the various PowerPC 405 features.

As we show in Table **??**, the instruction and data caches together account for more than 95% of the 405's sensitive bits. This is compounded by the fact that the PowerPC 405's cache parity circuit contains a known hardware error that prevents its use [**?**]. Without the parity circuit enabled, the PowerPC 405 does not correct cache parity errors resulting from SEUs. Consequently, the caches become a critical element for fault injection.

IV. MSIS Architecture

The Memory Sentinel and Injection System (MSIS) is a fault injector for the PowerPCs 405(s) in Xilinx Virtex-II Pro and Virtex-4 FX FPGAs. The MSIS introduces software faults to an application by flipping bits in the processor general purpose registers, special purpose registers, or the instruction or data caches. When a fault is injected, its details are logged so post injection analysis can be performed to determine the cause of a failure. The MSIS also uses the FPGA fabric to monitor the bus transactions generated by the PowerPC. The monitor ring can modify a bus transaction to create a data difference between memory and the processor in order to emulate an SEU within the cache, or it can act as a memory sentinel by preventing an illegal bus transaction from writing to read-only memory. This dual functionality allows the MSIS to remain in a system from the testing phase, to a beam test, and into the field. In addition, the monitor ring can be protected by TMR and scrubbing because it is implemented in the configurable fabric.

The MSIS injects an error into the processor by performing the following steps:

1) Use FPGA logic to assert an interrupt
2) Decide on a bit flip location
3) If required, setup FPGA logic for bit flip
4) Log the type of injection
5) Perform a bit flip
6) Restore the processor to its pre-interrupt state

To perform these steps, the MSIS is divided into a software portion (SW-MSIS) and a hardware portion (HW-MSIS) implemented in the FPGA fabric (see Figure **??**).

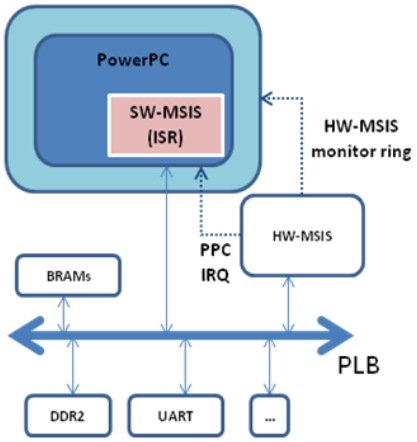

Figure 2. The MSIS is divided into the SW-MSIS and the HW-MSIS. The monitor ring helps to corrupt data going to cache and protect memory from illegal accesses.

The HW-MSIS triggers the SW-MSIS with an interrupt, and the SW-MSIS performs the necessary steps to flip a bit within the running software application. The following sections describe the SW-MSIS and HW-MSIS in detail.

A. Software MSIS

As a part of the FPGA design, the HW-MSIS is connected to the critical interrupt of the PowerPC 405. When the interrupt is triggered, the PowerPC jumps to a fixed address in the interrupt vector table, which has 256 bytes defined for the interrupt service routine (ISR) [**?**]. The GCC compiler provided by Xilinx creates a fixed set of instructions at this interrupt vector. These instructions push all of the general purpose registers (GPRs) and a limited number of the special purpose registers (SPRs) on to the stack, called the vector stack, before calling a user-defined ISR. The compiler also pushes the nonvolatile registers (r14–r31) onto the ISR stack upon entering the ISR. When the ISR returns to the vector table, the processor registers are returned to their pre-interrupt state by popping the data off the vector and ISR stacks (see Figure **??**).

When an interrupt is triggered by the HW-MSIS, the injection process starts by creating the vector stack and the ISR stack. The creation of the vector stack and the ISR stack may pollute the instruction and data caches. For example, if the interrupt vector routine is in a cacheable section of memory, up to 8 instruction cache lines of 512 total (1.56%) may be modified. Similarly, the interrupt vector table also stores the registers (160 bytes) to the stack, which is generally cacheable. If the stack pointer is not cache line aligned, these memory writes may modify up to 6 lines of 512 total (1.17%) in the data cache. We are investigating the use of a custom interrupt vector table as a way to further reduce cache pollution. Note however, that the full contents of the cache may be modified by the MSIS.

The first step of the SW-MSIS ISR is to disable cache line allocation on load and store instructions. With this feature disabled, reads and writes to memory should no longer impact the contents of the cache memories. Thus, the SW-MSIS can maintain a level of application transparency by preserving the majority the cache contents. The next step is to read from the pseudo-random number generator (PRNG) in the HW-MSIS to determine the type of fault to be injected: GPR, SPR, instruction cache, or data cache. The type of injection is chosen randomly according to a user-defined probability distribution function (we currently use a uniform distribution) over the sensitive bits in the processor (see Table **??**). Then, another read from the PRNG determines the bit to flip in the chosen location. The bit flip is completed by performing an XOR of the chosen bit in the chosen location. Once the corruption is completed, the allocation of cache lines is re-enabled and the ISR returns.

1) General Purpose Registers: The general purpose registers (GPRs) are all stored in the vector stack when the ISR is triggered. So, if the ISR flips a bit in a register while the ISR is running, it will be restored to its original state when the vector stack is popped back into place. Therefore, the GPRs need to be modified at their location in the vector stack. These offsets are fixed as a part of the GCC vector table assembly instructions. So, we have created a set of macros to use as offsets into the vector stack for each register. These macros allow us to modify the GPRs in their location on the vector stack. Then, when restored from the vector stack, the application will now have a one bit difference in the selected register.

The only GPR that this philosophy does not apply to is Register 1 (r1). Register 1 is used by GCC as the stack pointer. The interrupt vector table assembly code restores r1 to its original state by adding an immediate value to it instead of reading it back from memory. Since the compiler knows how many bytes it pushed on to the stack in the vector table, it uses an add instruction as an optimization to pop all of the data off the stack at once. So, to corrupt r1, we needed to modify the vector table assembly code. We have added an additional instruction to XOR an immediate one-hot value into r1 after the add instruction but before the RFI instruction. When the stack pointer is chosen for injection, the immediate value in this instruction needs to be modified to corrupt one of the bits in r1.

2) Special Purpose Registers: Bit flip injection to the Special Purpose Registers (SPRs) is similar to the GPR injection. Some of the SPRs get stored on the vector stack, again in known locations, and some do not. The registers on the vector stack are corrupted the same way as the GPRs. The SPRs not stored on the vector stack can be modified in their actual processor locations. However, since we rely on an interrupt to start the SW-MSIS, we cannot inject a bit flip that will disable the critical interrupts in the processor. All other SPR injections are legal changes.

3) Instruction and Data Cache: Both the instruction and data caches are 16 KB, 2-way set associative with a LRU cache replacement policy. Each cache has a parity detection circuit. However, as discussed above, these parity circuits will cause an illegal exception when enabled. Therefore, we need to be capable of flipping the bits in both cache memories.

From the processor's perspective, the instruction cache is read-only. As a result, to modify a bit in the instruction cache, it needs to be corrupted entering the processor. We use the HW-MSIS monitor ring to perform the bit flip (See Figure **??**). One advantage of this approach is that the corruption will only last as long as the cache line is valid. When the cache line is invalidated and reloaded from memory, the correct bits are loaded into cache. So, while the corrupted cache line is active in the processor, it will behave as if it has been hit with an SEU.

When the SW-MSIS selects an instruction cache corruption, the allocate cache line on load and store processor feature is re-enabled. Then, the address and data to corrupt are loaded into the HW-MSIS. The selected cache line is invalidated and then touched. As the data passes from memory to the PowerPC, the data word at the selected address is replaced with the corrupted data word. This procedure has been validated, by reading the instruction cache contents before and after the bit flip to observe the change in data.

The data cache injection is similar to the instruction cache. Since writes can occur to the data cache, an injection to the data in the data cache can occur in place. However, if the data cache line is valid but not dirty, the cache line needs to be invalidated and re-touched into cache so it can be corrupted by the monitor ring.

Injecting into the cache flags is accomplished similarly. Space does not allow us to describe the injection strategies for all cache configurations; however, most flags may be inverted through invalidating and/or touching/retouching the target or neighbor cache directions using the existing PowerPC processor instructions.

B. Hardware MSIS

The HW-MSIS is responsible for supporting the SW-MSIS in performing a bit flip and for protecting the read-only sections of the running application. The HW-MSIS provides a timer, a pseudo-random number generator (PRNG), and an address/word pair for data corruption to support the SW-MSIS.

1) Fault Injection Support: The HW-MSIS contains a one-shot 64-bit timer to trigger an interrupt to the PowerPC. When the timer expires, the SW-MSIS performs a bit flip within the processor. The timer can be enabled, disabled, or reloaded through software.

The PRNG is used by the SW-MSIS to decide on the type and location of a fault injection. It provides a software

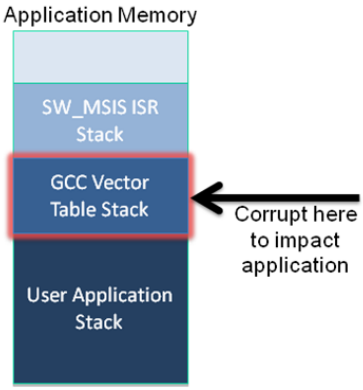

Figure 3. When the MSIS causes an interrupt, the vector table saves the application state on the stack and then calls the interrupt service routine.

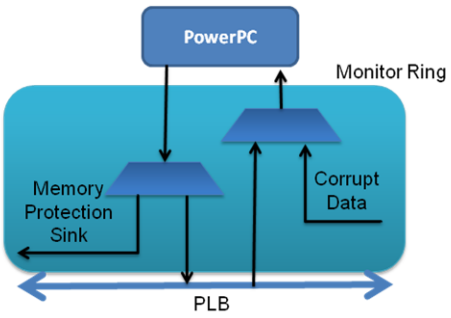

Figure 4. The MSIS monitor ring can protect memory from illegal transactions and corrupt data on its way to the PowerPC.

interface to load a new seed value, to free-run, and to only generate a single random number for each request.

The address/data word pair is used by the SW-MSIS to corrupt a data word as it passes from memory to the PowerPC. When this data corruption is enabled, the data word is placed on the bus read data lines when the address has been identified. This feature is used to corrupt either the instruction or data caches.

2) Memory Protection: In addition to corrupting the bus data as it passes from the memory to the processor, the MSIS monitor ring can also provide a level of protection to an application by preventing illegal writes to read-only memory spaces. A read-only memory space is specified to the HW-MSIS in one of two ways. First, a configurable number of address ranges are provided as non-resettable values. These memory ranges will not be changed or invalidated on an FPGA reset. This type of address range is useful for an application that is pre-loaded into the block RAMs of an FPGA, like a boot loader. The second type of memory space is specified by the user. A configurable number of low and high address pairs, stored in registers on the FPGA, are provided to allow an application to specify any further read-only memory sections. Then, if a memory write occurs to any of these locations, the transaction is stopped by the monitor ring and an interrupt is generated to the processor (see Figure **??**). A write to read-only memory should cause a failure because we know something unexpected has happened to the application. When the interrupt is identified, the PowerPC then needs to decide on a course of action.

Using these techniques to inject bit flips, we are able to emulate a fault in the vast majority of sensitive bits within the PowerPC. Since the intended target of the MSIS is a processor embedded within a Xilinx FPGA, it has been integrated with the standard tool flow to make it usable in any design. The MSIS can also be targeted at the PowerPC 440 in a Xilinx Virtex-5 FX FPGA or at the Xilinx MicroBlaze soft-core processor with minimal effort. These processors use a similar bus structure so the changes to the HW-MSIS would be minimal.

V. Application and Testing

In order to evaluate the MSIS, we have developed a synthetic test application modeled after a pair of space-based scientific applications. The application is composed of computational kernels from both hyperspectral and synthetic aperture radar (SAR) imaging. From hyperspectral we borrow a representative thresholding kernel, and from SAR we borrow the complex multiply and FFT kernels. Data sizes are kept small (size 128 FFTs) in order to execute entirely in block RAM, yet will still turn over cache contents. The small data sizes aid in making the results easier to analyze but is not a requirement of the MSIS. All general purpose registers are used within the test application as are both caches. Most special purpose registers are not referenced directly within the application. However, manipulating the SPRs at runtime often results in undesirable side effects. For example, disabling/enabling cacheable regions, debug modes, and interrupts. No operating system was used in our tests. Further, these results do not leverage the MSIS memory sentinel in order to focus this work on MSIS injection.

The application repeatedly performs 1-dimensional FFTs and complex multiplication followed by thresholding in order to mimic both hyperspectral and SAR imaging. At system startup a golden output is calculated that is used to verify results during the injection campaign. A backup of the golden output is also maintained in order to ensure the accuracy of the golden output throughout the injection campaign. If at any time a data error is found (either the golden outputs or a computed result), the PowerPC logs the error to the UART, resets itself, recomputes golden outputs, and continues the injection campaign.

At startup, the application completes a calibration phase where an average execution time is derived. The execution time is used by the MSIS to provide an upper bound on the execution during which the MSIS may inject an error. After calibration, the application enters the injection phase,

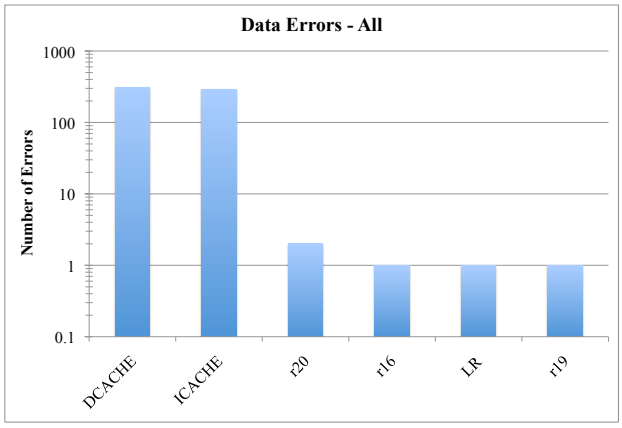

Figure 5. Summary of injections resulting in data errors.

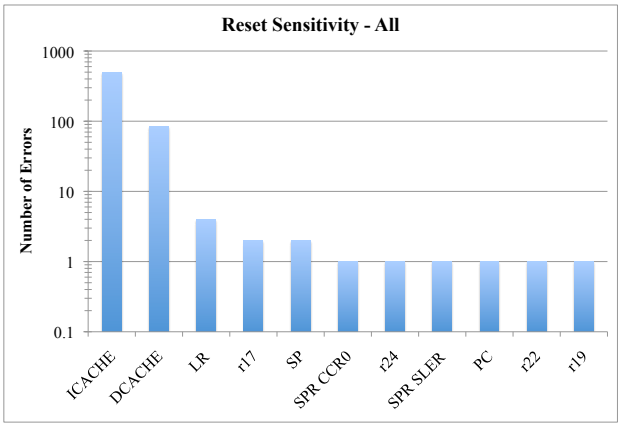

Figure 6. Summary of injections resulting in resets.

Table III
SUMMARY OF FULL INJECTION RESULTS

Result	Total	Percent
Good Data	8798	88.0%
Data Error	609	6.1%
Reset	593	5.9%

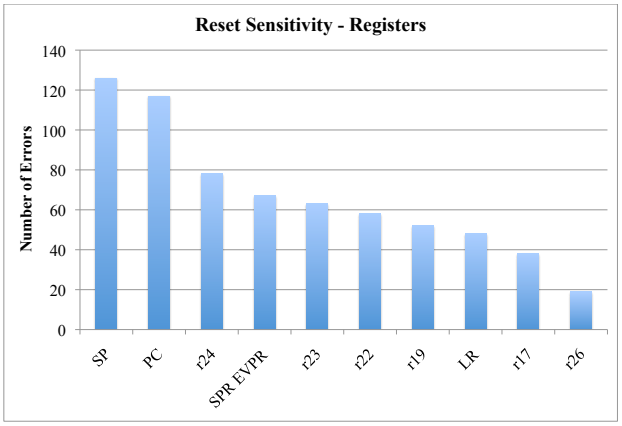

Figure 7. Top registers responsible for resets.

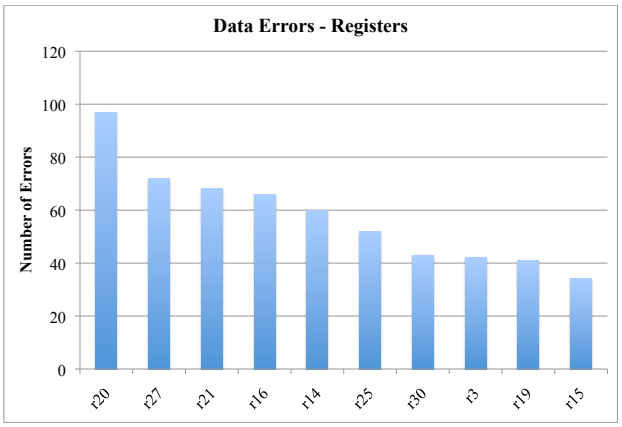

Figure 8. Top registers responsible for data errors.

which consists of the algorithm repeatedly executing and validating within an infinite loop. The SW-MSIS interrupts the processor and injects an error into the test at a random clock cycle within the bounds of the execution time derived during the calibration stage. At the end of each trial, the PowerPC under test writes the test results to the UART which is logged to a local file system.

Considering that even the simplest applications execute billions of instructions, the result of bit error injections may manifest in a huge variety of ways. In this paper we do not claim to have performed a comprehensive evaluation of the susceptibility of the PowerPC 405. Such an evaluation would require orders of magnitude more injections than we present in this paper, and would likely remain application-specific. Instead, we seek to provide a tool that researchers may use to evaluate their own applications in an automated and straightforward fashion.

Nevertheless, we present two injection campaigns designed to confirm that the PowerPC 405 behaves as expected, given bit error injections. We cannot predict the behavior of an arbitrary injection - indeed many injections will simply not propagate to the application level. Some injections, however, are quite predictable. For example, manipulating the stack pointer or program counter are extremely likely to put the processor into an undefined state (i.e. hang the processor). Other injections, such as cache injections, are far more subtle and depend on program execution – whether the cache values were consumed or simply evicted, for example.

A. *Injection Results*

In order to demonstrate the MSIS we have completed two injection campaigns of 10,000 injections each. In the first campaign we allowed the MSIS to inject into each writable GPR and SPR as well as the data cache and instruction

Table IV
SUMMARY OF REGISTER INJECTION RESULTS

Result	Total	Percent
Good Data	8243	82.4%
Data Error	913	9.1%
Reset	844	8.4%

Table V
BREAKDOWN OF DATA ERRORS AND RESETS BY BITS.

Bit field	Description	Data Errors	Resets
		Total (per bit)	Total (per bit)
0–5	Opcode	38 (6.3)	101 (16.8)
6–10	Destination Register	51 (10.2)	44 (8.8)
11–31	Other Fields	184 (8.7)	261 (12.4)

cache. Because the cache sizes are overwhelmingly larger than the sum of all of the registers, the vast majority of these injections targeted the caches. To gain a better understanding of the impact of register-based SEUs on the application, we then performed a second test of 10,000 register-only injections. This allowed us to more accurately test the impact of register bit flips independent of the caches.

Test results may fall under three categories: "Good data", meaning the test ran to completion without any errors propagating to the application-level; "data error", meaning that a data error was found during the comparison step; and "reset", meaning that the processor entered an unknown state and required a restart.

In Table **??** we present a high level overview of our full injection campaign results. Predictably, the results show that, overall, the vast majority of injections have no impact on the running application. Nevertheless, it's clear that certain processor elements are more sensitive to upsets than others. For example, of the 10,000 injections recorded 4973 were into the data cache, and 4921 were into the instruction cache. Examining Figure **??** we see that data error sensitivities are spread evenly over both the data cache and the instruction cache. However, examining Figure **??** shows a much different picture where injections into the instruction cache are nearly 6 times more likely to result in a reset than an injection into the data cache. This is reasonable given the characteristics of our test application: small code size that is only twice the size of the cache.

Nevertheless, if we further examine those instruction cache injections that caused either a data error or a reset an interesting pattern emerges. In Table **??** we present the breakdown of data errors and resets by the bit field each injection modified. We also provide the normalized per-bit data for each field as well. The PowerPC 405 reserves bits 0–5 for the instruction opcode and bits 6–10 for the destination register (with the single exception of the unconditional branch), while bits 11-31 depend on the instruction. From Table **??** we can clearly see that injections which modify the opcode are more than 2.5 times as likely to result in a reset than a data error. This is expected as a single bit modification to an instruction's opcode may drastically change the instruction that is executed. Bits 6–10 initially do not appear to heavily favor either resets or data errors. However, examining the normalized data reveals that the destination register is the most sensitive to data errors. Intuitively, this makes sense as manipulating an instruction's destination register may easily affect data at the application-level. Bits 11–31, however, favor resets over data errors. These bits have different meanings depending on the instruction being executed. In most cases bits 11–15 encode a source register, leaving bits 16–31 remaining. The remaining bits are split primarily between immediate values, secondary source registers, and extended opcodes. The most common case uses bits 16–20 for a secondary source register, and bits 21–31 for the extended opcode. The extended opcodes are particularly vulnerable because they present a relatively wide target.

Because the caches represent over 90% of the injections described in Table **??**, the register injections that occurred during this campaign are dwarfed by the number of cache injections. In our second injection campaign we focused entirely on the registers. In Table **??** we present our high level register injection results. We can see that register injections have somewhat higher data error and reset rate, though like the cache injection campaign, data errors and resets are split evenly with a slight preference towards data errors.

In Figure **??** we present the top 10 registers associated with processor resets. Predictably, the program counter, stack pointer, exception vector prefix register (EVPR), and the link register are most prevalent. Specifically, the stack pointer and program counter resulted in resets nearly every time they were modified: 95% and 80% for the stack pointer and program counter, respectively. Notably all general purpose registers in Figure **??** represent nonvolatile registers according to the PowerPC embedded Application Binary Interface (e-ABI) [**?**]. These registers must be preserved (saved and restored) during function calls. Consequently, it's no surprise that injections to these registers often result in observable application-level errors.

In Figure **??** we see a similar result. However, unlike the resets which were dominated by special purpose registers such as the stack pointer and program counter, data errors are caused almost exclusively by the nonvolatile registers (r14–r31) described above. Interestingly, there is little overlap between the top data error producing registers and the top reset producing registers (the single exception being r19). We believe that this is an artifact of both our application and the compiler's (GCC) register allocation strategy. The only volatile register represented in Figure **??** is r3. According to the ABI, r3-r4 are used for both parameter passing as

well as return values. Our test application makes heavy use of both function calls and function return values for both complex multiplication, trigonometric functions, and the soft floating point library. Register 4, while not represented in the top 10 sensitive registers, is highly sensitive to injections as well. It accounts for 29 data errors. We speculate that r3 is more sensitive than r4 due to the heavy use of single precision floating point throughout the application, while r4 is used primarily for return values from the complex multiply functions.

While these injections do not represent a comprehensive evaluation of the PowerPC 405, for example, we cannot target the execution pipelines or any external buses, they do align well with our expectations of the processor given the injections observed. This suggests that the MSIS is reliably injecting bit errors into the test application.

VI. Conclusions and Future Work

In this paper, we have introduced the Memory Sentinel and Injection System (MSIS), a tool for fault injection into the PowerPC 405. The MSIS introduces faults by inverting bits in the register sets (general and special purpose) as well as the instruction and data caches. Through a series of 20,000 injections we have demonstrated the PowerPC 405's behavior in the presence of register-based and cache-based upsets. Given known sensitive locations, such as the program counter and stack pointer, we showed that the PowerPC fails in an expected fashion.

In the near term, we will begin to inject faults into the fabric of the FPGA, while running our test application, to determine how FPGA configuration upsets can impact the performance of the embedded processor. Moving forward, we intend to compare the results of this paper with the results from a radiation beam test running the same application. Performing this comparison will further validate the MSIS as a method for testing the reliability of software in the presence of failures.