A Parallelized Neutron Radiation Testing Technique to Understand Failures Within a Complex SoC

Jeffrey Goeders, Weston Smith, Maria Kastriotou, Michael Wirthlin

***Abstract*—Complex System on Chip (SoC) are increasingly used in embedded applications where both high computing performance and low power consumption is required. These devices are especially important in embedded environments where safety and reliability is critical, such as Advanced Driver Assistance Systems (ADAS) and space applications. The failure modes and failure rates of these devices due to single-event effects are not well understood and methods for testing these devices is tedious and time-consuming. This work introduces a methodology for simultaneously testing multiple components of a system-on-chip (SoC) device. This methodology is used to evaluate the failure modes of several components within an AMD UltraScale+ MPSoC device under neutron radiation. The results from two neutron beam tests are described and compared.**

***Index Terms*—single event upset, system-on-chip, neutron testing**

I. Introduction

System on Chip (SoC) devices that include multiple processors, dedicated memory controllers, integrated memories, and high-bandwidth I/O provide high computing performance at relatively low power consumption. The computational efficiency of these devices is obtained by integrating multiple components into a single package and using the latest processor technology. These devices are especially important in embedded environments where safety and reliability is also critical autonomous vehicles, control systems, and space applications.

While radiation-hardened Integrated Circuit (IC) are available, they are often expensive and have much lower performance than modern Commercial-off-the-Shelf (COTS) devices. Given this, there is a growing interest in using COTS devices in space applications, especially modern System on a Chip (SoC) devices [1]–[4], which can provide multiple processors, memory controllers, hardware accelerator engines, and other Input/Output (IO) controllers all within a single IC package [5].

While COTS SoCs provide the performance and low power consumption needed for embedded applications, they are typically not designed to operate in high-radiation environments, and are therefore susceptible to radiation-induced failures. The radiation-induced failure modes of the components and processors within these complex devices is not well understood. This limitation means that COTS SoCs must be tested to understand how they will perform in a given environment, and to identify any failure modes that may occur.

Understanding the failure modes of modern SoC devices and qualifying them for a harsh radiation environment is challenging. First, the devices are complex and involve many cores, custom intellectual properly (IP) blocks, and components that each have their own unique failure modes and failure rates. Each component or core may need a custom test to directly exercise and identify these component-specific failure modes. Second, the components of the SoC are highly interconnected and failures in one component may cause other components to fail. Comprehensive tests must exercise such inter-component activity to identify these failure modes. Third, the per-bit cross-section of modern SoC devices is very low making it very difficult to identify individual component failures [6]. Increasingly more radiation fluence is needed to obtain the statistically significant error rates when using traditional testing methodologies.

This work presents an SoC testing methodology that seeks to address these challenges by performing multiple tests on multiple cores and components simultaneously to maximize the collection of failure data. This approach is an extension of our previous work [7], where we presented a testing methodology using a single-core round-robin strategy to monitor the SoC components for failures. In this work we parallelize the testing across multiple cores, and strategically assign the tests to cores to increase the number of errors that can be observed in a given amount of time. We demonstrate this methodology with neutron radiation testing of the Xilinx UltraScale+ MultiProcessor SoC (MPSoC) device at the Los Alamos Neutron Science Center (LANSCE) and the ChipIR facility within the UK ISIS Neutron and Muon Source facilities. Although this work was performed using neutron radiation, this methodology is general enough to be used for other radiation sources, such as heavy ions or protons.

II. Related Work

A. Processor and SoC Testing

Radiation testing of processors is an active research area with decades of academic, industrial, and government testing efforts. Testing has been performed on a very wide range of processors, including, as some examples, Intel Pentium MMX and Pentium II processors [8]–[10], IBM PowerPC [11],

Jeffrey Goeders, Weston Smith and Michael Wirthlin are with the Department of Electrical and Computer Engineering, Brigham Young University, Provo, UT, USA. They can be contacted at jgoeders@byu, wms29@byu, and wirthlin@byu, respectively. Maria Kastriotou is with the ISIS Facility, U.K. Research Innovation (UKRI)-Science and Technology Facilities Council (STFC), Swindon, U.K. She can be contacted at maria.kastriotou@stfc.ac.uk.

This project is sponsored by the Department of the Defense, Defense Threat Reduction Agency under award HDTRA1-20-2-0002. Experiments at the LANCE facility were supported by the DOE under Proposal NS-2022-9138-A and experiments at the ISIS Neutron and Muon Source were supported by a beamtime allocation RB2310429 from the Science and Technology Facilities Council.

Cell/Opteron [12], RISC-V [13], [14], Freescale P2041 [15], Xilinx Zynq [16], Xilinx MPSoC [17], Xilinx Versal [6], Snapdragon 820 [18], and various ARM and Texas Instruments microcontrollers [19].

There are many different methodologies for testing processors, and the methodology used often depends on the goals of the test, the capabilities of the Device Under Test (DUT), and capabilities of the testing infrastructure. The most common method for testing processors is to execute a compute-intensive and/or memory-intensive benchmark on the processor with known golden output, and monitor the output (possibly over a Universal Asynchronous receiver/transmitter (UART)) for errors. For example, many of the frequently used benchmarks include matrix multiplication [6], [14], [15], [17], [20]–[27], Advanced Encryption Standard (AES) [6], [17], [19], [20], [24], [26], [27], quick sort [6], [20], [23], [27], Secure Hash Algorithm (SHA) [22], [23], Cyclic Redundancy Check (CRC) [14], [23], [25], linpack [12], coremark [19], vector summation [12], [13], [21], Fast Fourier Transform (FFT) [6], [13], and various memory fill and check tests [6], [8]–[11], [17], [19], [27].

Executing a single benchmark on a processor is a useful way to understand how the system may behave under a specific computational or memory-intensive load, but it does not supply much information regarding how individual pieces of underlying hardware respond to radiation. With modern processors typically being included as part of a complex SoC, there are an increasing number of ways that radiation can affect the full system and cause it to fail. If one knows ahead of time the specific components that will be relied upon in a deployed system, a test methodology could be developed to specifically exercise and target those components. However, in many cases, we are interested in a broader understanding of how the entire SoC will behave in a radiation environment, in which case a more comprehensive testing methodology is needed.

Several works have used testing methodologies that expose more information about how upsets and errors occur in a system, including tracking how individual SoC components respond to radiation. In some cases, these works have focused on writing test software that specifically exercises certain components of the device. For example, in [8]–[11], the authors use assembly instructions to exercise the processor's Arithmetic Logic Unit (ALU) and Floating Point Unit (FPU) units. Given that processor caches often represent a large radiation target, several works have focused on testing the caches, including [8]–[11], [15], [17], [27]. In [28], the authors specifically create software that exercises and tests the ARM Neon Single Instruction, Multiple Data (SIMD) vector engines.

Another approach to test processors is to add instrumentation to the system that can monitor the internal state of the processor and detect errors. For example, in [20], the authors create custom hardware implemented in the field programmable gate array (FPGA) fabric that connects to the ARM debug ports and monitors the processor state. Other approaches have leveraged duplicate execution with comparison to detect errors, either through multiple processors cores [24], [26], or duplicated execution on a single core [22], [25], [27]. These types of approaches have typically been used to gain more observability into the processor state during an error, particularly to understand how error mitigation strategies may have failed, but are not necessarily focused on a comprehensive testing methodology for a complex SoC.

One of the most comprehensive examples of a testing methodology for modern SoCs is the System Validation Tool (SVT) developed by Xilinx [29]–[31]. This proprietary tool runs randomized test vectors through the system, with the goal of exercising many components of the SoC. The authors demonstrate this on both the Xilinx MPSoC [29], [30] and the Xilinx Versal [31]. The authors mention that this tool exercises the "APU, RPU, GE, SD/eMMC, CAN, UART, GPIO, SysMon, CSU, PMU, SATA, LPD Switch, Central Switch, LPD-DMA, FPD-DMA, GPU, SMMU/CCI, PSGTR and DDRC" components of the device [31]. However, the goal of this tool is not to assess the source of errors in the system, or to understand how individual components respond to radiation, but rather to provide a single cross-section of the device as a whole. In addition, the tool itself is not publicly available, and details of the internal workings of the tool, such as utilization percentages for different SoC components, are not available in the literature.

In contrast, this work focuses on a methodology for testing many SoC components in parallel, with the goal of understanding how individual components respond to radiation, and how they may fail. We demonstrate this methodology on the Xilinx UltraScale+ MPSoC device, but are currently working on applying the same methodology to other SoC devices, which we plan to test and report on in the near future.

B. Xilinx MPSoC Testing

In this work we perform radiation testing of the Xilinx/AMD Ultrascale+ MPSoC chip [32]. This chip was chosen since it is a modern SoC with many different components, which we have used in previous work [7]. The device is a 16nm FinFet chip with four ARM Cortex-A53 Application Processing Units (APUs), two Cortex-R5F Real-time Processing Units (RPUs), L1 and L2 caches, several RAMs, FPGA fabric, and many specialized hardware modules, such as vector and cryptographic units.

Aside from the previously mentioned Xilinx SVT tool, several other works have already performed radiation testing on the Xilinx UltraScale+ MPSoC device. Anderson et al. [17] also tested the device at LANSCE. Their work focused on testing the ARM Cortex-A53 processors by executing AES and matrix multiplication executables, as well as tests that would check for upsets in the On-Chip Memory (OCM) and processor caches. This work used separate executables to test each component, and unlike this work, did not focus on a methodology for testing multiple components simultaneously. Finally, Hiemestra et al. [33] tested the MPSoC device with proton irradiation, and Lee et al. [34] tested it in heavy ion and neutron environments; however, both works focused only on the FPGA fabric of the device, and did not test the SoC processing system.

C. Our Previous Work

In [7] we presented an SoC testing methodology with similar goals to this work, namely to understand per component error rates in a complex SoC device, and used the same DUT as this work, the Xilinx UltraScale+ MPSoC device. This earlier work used a purely round-robin strategy to monitor the SoC components for errors, with each component test executing in sequence on a single core of the SoC. In contrast, this work presents a more complex methodology that leverages the multiple cores of the SoC to test multiple components simultaneously, and strategically assigns the tests to cores to increase the number of errors that can be observed in a given amount of time.

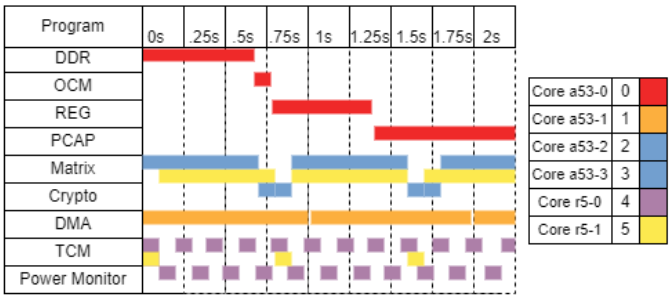

Fig. 1: One test configuration used in our radiation test experiments. Although not shown, the PCAP modules takes 40 seconds of execution, after which the *a53-0* modules repeat.

III. Parallel Test Methodology

The general approach of our testing methodology is to exploit the existence of multiple cores within the SoC by running multiple test programs simultaneously, each exercising a different component, or set of components, of the SoC. The ideal implementation would consist of having a separate executable program for each component, and run them in sequence, avoiding any interference between the programs (as was done in [17]). Unfortunately, this approach is time consuming, and requires significant beam time at a radiation facility, which can be expensive and difficult to schedule. This is especially problematic as certain SoC components have very low failure rates and may require days of testing to expose failure events [6]. For example, in [17], the authors tested the same device used in this work for several days, and never encountered an error running a matrix multiplication benchmark on the A53 processors. Rather than testing each component separately, we propose a parallelized approach where components are tested simultaneously.

A. Component Categorization

In deciding how to aggressively parallelize the testing process, we must first consider the various components of the SoC, and how interactions with the components affect their susceptibility to radiation-induced failures.

- *Passively accumulating and observable:* We recognize that certain components of the SoC will accumulate bit-flips over time, which can be easily observed, and occur regardless of whether or not the associated self-test application is actively using the component. This is primarily true for memory components, such as the Double Data Rate (DDR) Synchronous Dynamic Random Access Memory (SDRAM), OCM, Tightly Coupled Memorys (TCMs) and the FPGA fabric Configuration Random Access Memory (CRAM).
- *Errors exposed by active use:* Other components of the SoC have a hidden state that cannot be easily observed, and bit-flips may become masked and unnoticed if the component is not actively being used. For example, to determine error rates of processor cores, the standard test methodology is to run a compute-intensive benchmark, such as matrix multiplication. This ensures that the internal state is in use during the test, and any bit-flips that occur will more likely be propagated to produce errors in the output. For example, if a Central Processing Unit (CPU) were not utilized during radiation, and a bit-flip occurred in a data register, it would go unnoticed, as the unused data would be overwritten once a program began execution. In our testing, we classify the processor cores, the hardware accelerated cryptographic engines, and the Direct Memory Access (DMA) engines as components that usually require active use to expose upsets.

It is worth mentioning that the above classification is a helpful way to categorize component behavior to optimize the testing process, but in reality components may exhibit behavior that does not fit neatly into one of the above categories. For example, even when not in use, a processor or DMA engine may still accumulate bit-flips in certain sensitive internal state (such as configuration registers), that may cause failures when the component is used at a later point. However, we assume these cases are relatively rare, and that the above categorization is a useful approximation for optimizing the testing process. In addition, we believe that by treating components such as the CPU or DMA engines as requiring active use to expose errors, we are operating in a conservative manner, and will still expose upsets that may accumulate in these components even when not in use.

B. Test Partitioning

Given the above categorization, we can strategically allocate the self-test programs to different processing cores, in a matter that increases the number of errors that can be observed in a given amount of time. The general strategy is to maximize the amount of time spent running self-test programs on components that require active use to expose errors, and de-prioritize checking components that passively accumulate errors.

The DUT has six total CPU cores: four higher-performance 64-bit ARM A53 cores, and two lower-performance 32-bit ARM R5 cores. Certain components of the SoC are only accessible by certain cores, which adds an additional constraint to the test partitioning. For example, each R5 core has its own TCM that is not accessible by other cores, and only the A53 cores contain caches and cryptographic engines.

Figure 1 illustrates one configuration we used for assigning test programs to cores during radiation testing. In this configuration, one A53 core (a53-0) is used to check the

passively accumulating components (highlighted in red: DDR SDRAM, OCM, bus registers, and CRAM, via the Processor Configuration Acess Port (PCAP)). Since they are passively accumulating, they can be checked at any time, in any order,and will largely be unaffected by longer delays between checks. Grouping them together in one core frees up the other cores to focus on other components that may require active use to expose errors.

In this configuration, the other three A53 cores are constantly utilizing the components that require active use to expose errors. Core a53-1 is used to test the DMA engines. Since there is only one set of DMA engines shared between all cores, it does not make sense to run this test on multiple cores. The remaining two APU cores (a53-2 and a53-3) repeatedly execute the matrix multiplication benchmark, which exercises the CPU cores, along with testing the cryptographic engines. These modules are the least error-prone, and are the most resilient to radiation, so they are run on the most cores to maximize the number of errors that can be observed.

On the R5 cores, one core spends most of it's time running matrix multiplication benchmark for general CPU core testing with occasional breaks to check its TCM for errors. The other core keeps close watch on the power usage of the DUT, watching for any spikes that may indicate a failure, while also checking its TCM for errors.

The goal of this approach is to produce as many errors as possible in hard-to-test components, such as the CPU cores, cryptographic engines, and DMA engines, while still checking the passively accumulating components at a reasonable rate. If we can obtain a statistically accurate error cross-section of these components that require active use, we can then use the percentage of time spent running this component to extrapolate the error cross-section in a scenario where this component was used continuously.

Unfortunately not all system components can be tested with this parallel approach. For example, in related experiments we have tested the cross-section of the processor caches; however, in order to test the caches no other test programs can be running, as they will interfere with the cache state, and cause false positives.

We recognize that testing multiple components in parallel could cause cascading effects between the components, and that the test programs could interfere with each other. We have attempted to mitigate this by creating test functions that are as isolated as possible; however, it is difficult to completely eliminate any cascading effects. Better understanding of these effects, and measuring how errors propagate between system components is interesting future work. In this work, we attempt to avoid these effects and measure the error rates of individual components before they propagate to other components. In general, our tests try to immediately detect and correct errors, rather than allowing them to propagate. In cases where errors cannot be corrected, the test is halted and the system is rebooted (as described in Section IV-A). While a parallelized test approach may increase the likelihood of cascading effects, we believe that the benefits of increased data collection will often outweigh the potential downsides.

C. Component Test Functions

Our radiation test configurations consist of multiple executables that are programmed onto the various cores of the SoC. These executables are written in C, operate in a "bare metal" environment, and consist of a main loop that continuously runs self-test functions for one or more components of the SoC. The tested components are not exhaustive, and the DUT contains many other components that could be tested. We have focused on the components that we believe would be most likely to exhibit failures (eg., large memories), although there are likely other components that have not been tested that could be more susceptible to radiation. It is not always easy to predict which components will be most susceptible to radiation; for example, we test the CPU cores with matrix multiplication since this is a common test performed in processor testing; however, as the results show in Section IV-D, the CPU cores are very resilient to radiation, and other components, such as the DMA, exhibit more failures. It is not the goal of this work to fully characterize the DUT, but rather to demonstrate a methodology for testing multiple components in parallel.

The following describes the self-test functions used in our test configurations. The test executables are compiled using *aarch64-elf-g++* version 8.2, with the default optimization level.

A53 APU cores: The A53 cores are tested by running a floating-point matrix multiplication operation. Two matrices of size 125x125, using the `double` data type, are filled with random data using a fixed seed during the start of the program. They are then repeatedly multiplied into a third matrix. All memory values of the resultant matrix are then XORed to produce a single hash value, which is compared against a pre-computed and hard-coded golden value. This operation is a compute-intensive task that will exercise the CPU hardware, including the ALU, FPU, caches, DDR memory controller, and more. While the function cannot distinguish between errors in these various parts of the system, the caches are Error Correcting Code (ECC) protected, and we have a separate test for the DDR memory controller. It should be noted that this CPU has been proven to be very resilient to radiation, as the work in [17] did not observe any errors in the CPU cores after several days of testing. A bug in the test software that prevented the entire resultant matrix from being checked was discovered after the LANSCE and 2023 ChipIR tests were completed. This bug was fixed, and the test repeated in a subsequent visit to ChipIr in 2024. As such, the results in Table I only contain this test from the 2024 ChipIR visit.

R5 RPU cores: The R5 cores are tested using the same matrix multiplication function as the A53 cores.

DMA Engines: The DUT contains 16 DMA channels. The test consists of initializing a 4MB section of memory with a fixed pattern for each DMA channel. Each DMA channel is then set to transfer its 4MB section of memory to another location. The code then waits for all DMA channels to complete, and then checks the destination memory for errors. While the DMA engines are quite

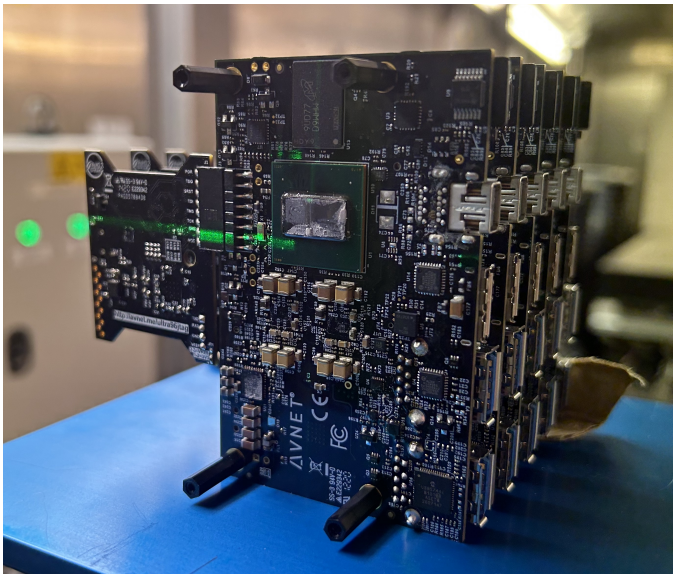

Fig. 2: Five Ultra96v2 boards at the Science and Technology Facilities Council (STFC) ChipIr test.

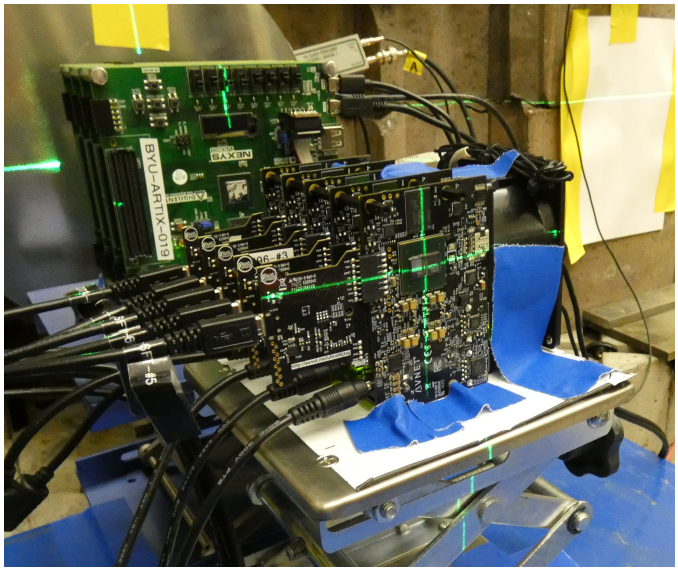

Fig. 3: Five Ultra96v2 boards at the LANSCE test. The boards between the Ultra96v2 boards and the beam emitter are from a separate experiment.

complex, with support for different transfer modes, such as scatter-gather, we are only testing the simple memory-to-memory transfer mode.

Cryptographic Engine: The DUT contains dedicated cryptographic hardware for performing AES encryption and SHA hash algorithms. These components were tested by filling a random array with known values, and then performing AES encryption on the array, followed by a SHA hash on the encrypted data. The hash was then compared against a pre-computed golden value.

OCM: The OCM is tested by initializing the entire memory with a fixed pattern, and then periodically checking the memory for errors.

TCM: The TCM is tested by initializing the entire memory with a fixed pattern, and then periodically checking the memory for errors. Each R5 core has its own private TCM, so this test is run on both R5 cores.

CRAM: The FPGA fabric is configured using a blank bitstream, and then the CRAM is periodically checked for bit flips. The PCAP interface provides access to the CRAM, allowing the processors to read the CRAM values and correct any errors.

DDR: A periodic DDR memory test is performed by writing and reading a known pattern to a reserved section of DDR memory. Any deviations in the read-back data are identified as DDR errors.

IV. Radiation Testing

A. Test Methodology

The test board was a Ultra96v2 board, containing a Xilinx Zynq UltraScale+ MPSoC device and 2GB of DDR4 SDRAM. Five boards were tested simultaneously, with different boards executing a different set of test programs (Figure 1 shows one such configuration). The boards were monitored by a single controller program running on the host computer, which would program the various executables onto the processor cores, and monitor the output of each board. The test programs were programmed onto the boards using the Xilinx Software Command-line Tool (XSCT), which provides Tcl commands to reset, program, and run executables on the SoC device via the Joint Test Action Group (JTAG) Universal Serial Bus (USB) interface. The regular programming process consisted of powering on the board, and then following the standard Xilinx process of programming the default bootloader, followed by the test executables for each core.

Each self-test program prints regular status messages via the JTAG UARTs, which are captured to file and post-processed to determine error rates.

Upsets to the memories (OCM, TCM, CRAM) occurred regularly, and the self-test software modules would report these upsets over JTAG UART, and then correct them. In these cases the boards were not rebooted, and the test program would continue running.

For other modules, such as the matrix multiplication processor tests and the cryptographic engine tests, errors were rare, and there is no way to observe or correct the underlying upset. In these cases the test software would report the error, and the controller program would power cycle and reprogram the board.

In other cases the board would time out and stop sending messages, or may send a malformed message. This would also be logged and the board would be rebooted.

B. Communication Architecture

A significant challenge in aggressively parallelizing the testing process is how to effectively communicate with the DUT and monitor the output of the various test programs. In our test infrastructure, we had five instances of our DUT board running simultaneously. With each DUT containing six

TABLE I: Test Results from LANSCE and CHipIr

Module	LANSCE			ChipIr		
	Errors	Fluence	Cross Section (+/- 95% range)	Errors	Fluence	Cross Section (+/- 95% range)
OCM	711	$4.57{\cdot}10^{11}$	$7.42{\cdot}10^{-16}$ ($6.87{\cdot}10^{-16}$–$7.98{\cdot}10^{-16}$)	1500	$5.82{\cdot}10^{11}$	$1.23{\cdot}10^{-15}$ ($1.16{\cdot}10^{-15}$–$1.29{\cdot}10^{-15}$)
TCM	678	$9.15{\cdot}10^{11}$	$7.07{\cdot}10^{-16}$ ($6.53{\cdot}10^{-16}$–$7.61{\cdot}10^{-16}$)	2567	$2.13{\cdot}10^{12}$	$1.15{\cdot}10^{-15}$ ($1.10{\cdot}10^{-15}$–$1.19{\cdot}10^{-15}$)
CRAM	3714	$4.57{\cdot}10^{11}$	$1.79{\cdot}10^{-16}$ ($1.74{\cdot}10^{-16}$–$1.85{\cdot}10^{-16}$)	9375	$5.82{\cdot}10^{11}$	$3.55{\cdot}10^{-16}$ ($3.48{\cdot}10^{-16}$–$3.63{\cdot}10^{-16}$)
Crypto	1	$9.13{\cdot}10^{11}$	$1.09{\cdot}10^{-12}$ ($1.09{\cdot}10^{-13}$–$6.13{\cdot}10^{-12}$)	2	$1.22{\cdot}10^{12}$	$1.64{\cdot}10^{-12}$ ($1.64{\cdot}10^{-13}$–$5.90{\cdot}10^{-12}$)
DMA	17	$4.57{\cdot}10^{11}$	$3.72{\cdot}10^{-11}$ ($2.17{\cdot}10^{-11}$–$5.96{\cdot}10^{-11}$)	65	$5.83{\cdot}10^{11}$	$1.11{\cdot}10^{-10}$ ($8.38{\cdot}10^{-11}$–$1.39{\cdot}10^{-10}$)
A53 CPU*	-	-	-	25	$4.23{\cdot}10^{12}$	$5.91{\cdot}10^{-12}$ ($3.83{\cdot}10^{-12}$–$8.70{\cdot}10^{-12}$)
R5 CPU*	-	-	-	21	$2.11{\cdot}10^{12}$	$9.93{\cdot}10^{-12}$ ($6.15{\cdot}10^{-12}$–$1.51{\cdot}10^{-11}$)

The *OCM*, *TCM*, and *CRAM* data points are per-bit upset rates (ie. cross-section is scaled to number of bits in the memory), while the other data points are error rates for the entire component. *The A53 and R5 CPU results are from a separate ChipIr experiment in 2024, due to a data collection error in the 2023 experiment.

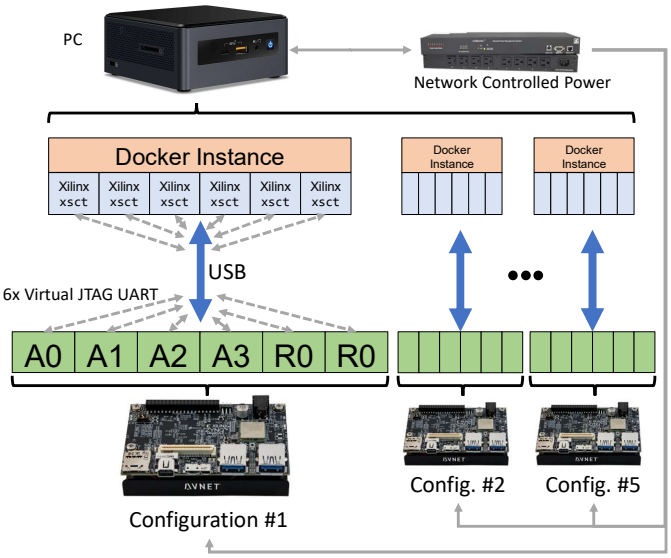

Fig. 4: Test setup

processing cores, this meant that we had 30 separate programs running simultaneously, each with their own output that needed to be monitored. Figure 4 illustrates the communication architecture we used to monitor the DUTs, which used a single host computer that captured the *stdout* of all 30 cores. While in previous experiments we have used a traditional UART to output the results of the test programs, sharing a single UART connection between the six cores of a DUT would require additional synchronization complexity. Instead, we opted to use the serial connection included with each core via the ARM debug port, which can be accessed via JTAG using the Xilinx *xsct* utility. This connection provides a virtual UART connection to each core, and allows the host computer to capture the output of each core separately and in parallel. The host computer was connected to each DUT via a single USB connection that was used for both configuration and monitoring of the virtual UARTs.

Due to limitations in the programmer interface of the development board we used (Avnet Ultra96v2), it is not possible to program individual boards when multiple are connected to a single host computer. We overcame this limitation by using docker to create a virtual machine for each DUT, and ran a separate instance of the Xilinx JTAG programmer in each virtual machine.

C. Test Facilities

We conducted three radiation tests, one at LANSCE at the Los Alamos National Laboratory in December 2022, and two at the ChipIr facility at the STFC Rutherford Appleton Laboratory in June 2023 and November 2024. The results in November 2024 were a repeat of the June 2023 test, in order to correct for a bug in the matrix multiplication test module.

At LANSCE, our experiment was run in the Weapons Neutron Research (WNR) Facility, using the 60R flight path, known as GEANIE [35][1]. The experiment consisted of four and half days of testing, during which our experiment received a total fluence of 1.8×10^{11} n/cm^2.

At STFC, our experiment was run on the ChipIr beamline, which provides a spectrum of neutron energies similar to what can be found in the atmosphere, but at a much higher flux. Figure 2 shows the test boards at the ChipIr facility. These experiments consisted of three days in 2023 and four days in 2024, during which our experiment received a total fluence of 7.1×10^{11} n/cm^2 and 1.3×10^{12} n/cm^2, respectively.

The actual neutron fluence for each experiment, and each test executable, was carefully measured by recording the net neutron fluence for each time frame that the board was executing the test code. Timestamps are included in the logs to identify the start and stop time of each experiment run, and are correlated with detailed neutron fluence data provided by the facility. The fluence and events were summed across all executions, and across all boards, resulting in some module tests receiving a higher fluence than others.

D. Test Results

Our methodology was successful in collecting a large number of error events from multiple components of the SoC in parallel. Several of the modules exhibited a very low cross-section, requiring a very large fluence to produce observable errors. Given the limited beam time typically available at these facilities, it would not have been possible to observe these events were it not for our parallelized testing approach.

[1]Note that the 60R flight path at LANSCE does not provide the same terrestrial neutron spectrum as does the 30R/30L flight paths found in the ICE/ICE II facilities. The neutron cross-section measured at this facility will be lower than that of the ICE/ICE II and ChipIR facilities.

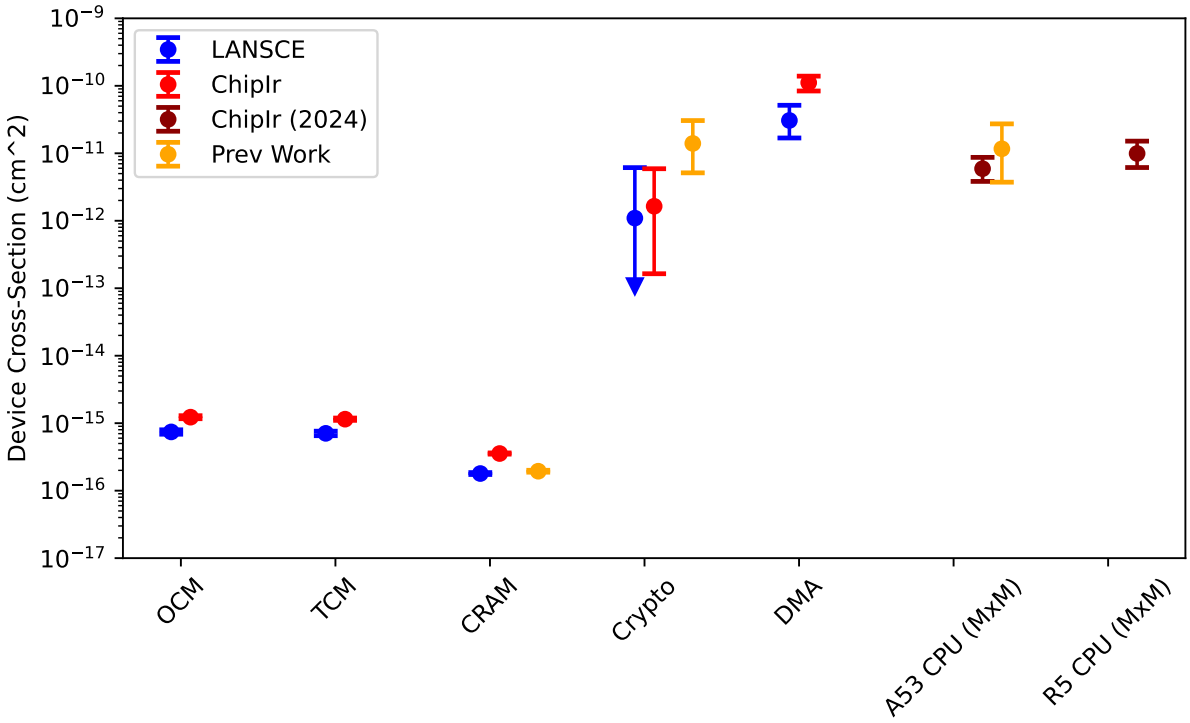

Fig. 5: Cross-section results of various SoC components. Some components encountered no errors during a given test, indicated by a downward arrow in the lower error bar. In these cases, the cross-section is computed assuming one error. The *OCM*, *TCM*, and *CRAM* data points are per-bit upset rates (ie. cross-section is scaled to number of bits in the memory), while the other data points are error rates for the entire component. The *Prev Work* values are from [7], where a sequential testing methodology was used.

Figure 5 shows the cross-sections for several of the SoC components that we monitored, with data from both the LANSCE test and the ChipIr test. Cross-section is calculated as the number of observed errors divided by the total fluence received by the component, summed across all boards and all test runs. In cases where the component is present on multiple cores and tested in parallel (eg. CPU, crypto, TCM), the fluence was summed across all cores. The fluence for each execution was calculated by correlating the start and stop times of the test program with the neutron fluence data provided by the facility. The start time is calculated as the time once all cores are running their test programs, and the stop time is calculated as the time when the test is stopped, or the board is rebooted (due to hang or error in certain modules as described in Section IV-A). While technically certain cores begin executing slightly before others (typically on the order of a few seconds or less), we do not account for this as tests typically run for several minutes, and this small difference in start time has a negligible effect on the fluence calculation.

For memory elements (TCM, OCM, CRAM), the cross-section represents the per-bit upset rate, while for other components (CPUs, cryptographic engine, DMA engines), the cross-section represents the error rate of the component as a whole, when performing the given task. The memory elements are a relatively large target, and many upset events were observed during the test, resulting in a statistically accurate cross-section. In contrast, the CPU and cryptographic cores exhibited a very low cross-section, with only a handful of events observed during the test. In some cases no errors were observed during the test, and the cross-section was computed assuming one error.

The results show fairly consistent error rates across the two test facilities, providing validation of our methodology. For the memories and DMA engines, the cross-sections are all slightly higher at the ChipIr facility. The facilities have a slightly different neutron energy spectrum, so this is not unexpected.

The cryptographic engines and processing cores exhibited very few events, so the cross-sections values are not as statistically accurate as the memory elements. It appears that the R5 cores are less susceptible to radiation than the A53 cores, as the R5 cores exhibited no errors, whereas the A53 cores exhibited four errors between the two experiments. However, with few events, the margin of error is quite large, and it is difficult to draw any firm conclusions.

No memory errors were observed by the DDR test during

either experiment (they are not included in the tabular or graphical results). This is not unexected since the radiation beam was applied to the SoC device and not the DDR memory. However, we anticipated seeing some errors in the DDR test due to upsets in the data in flight between the DDR and the processors through the DDR controller. The lack of DDR errors suggests that the errors encountered in the matrix multiplication tests are unrelated to the DDR or DDR memory controller.

The most interesting result we observed was the cross-section of the DMA engines. The results indicate that the DMA engines have about two orders of magnitude higher cross-section than the CPU cores. This is notable as it shows the importance of rigorously testing the SoC components; it is not enough to simply run a single compute benchmark (eg. matrix multiplication) on a CPU core and treat the failure rate as the cross-section of the entire device.

As discussed earlier in the paper, while we have attempted to isolate the test programs as much as possible, it is possible that errors in one component could propagate to another component. For example, since the CPU is executing all of these test programs, it is possible that an error in the CPU could appear as an error in any of the other test modules. However, given the very low error rate of the CPU (albeit on our limited matrix multiplication test), we believe this probably occurred very rarely. In most other cases we are testing specific hardware that is not shared between the test modules, so we believe that the errors we observed are likely due to the specific test module being run, and not due to errors propagating between modules.

E. Comparison to Previous Work Using Sequential Testing

In our previous work [7], we used a sequential testing methodology to test the same SoC device. Due to some issues with the test code, relatively few component types were tested in this previous work. Those that were successfully tested (and for which results have already been published in [7]) are shown in Figure 5 as *Prev Work* (the orange-color cross-section values). It should be noted that this previous work was performed at the LANSCE facility, but at a different beam line than the LANSCE results reported in this work, and thus the profile of the neutron spectrum was different compared to both the LANSCE and ChipIr results reported in this work.

When comparing the previous results to our new results, we see that the cross-section values for the CRAM are quite comparable between the two experiments, which is expected as Single Event Upsets (SEUs) in the CRAM accumulate over time, and we do not expect any difference in cross-section between a sequential and parallel testing methodology.

The Crypto and A53 CPU matrix multiplication test modules exhibit similar (overlapping 95% confidence ranges), but slightly lower cross-section values in these results, versus the previous sequential testing in [7]. In [7], the matrix multiplication test, and the cryptographic engine test, were run sequentially along with all of the other test modules, resulting in them being run periodically in the round-robin approach. In this work, as shown in Figure 1, the matrix multiplication and cryptographic engine tests were run almost continuously as the other modules could be offloaded to other cores. We would expect this to result in equal or possibly higher cross-section for these modules, as they are being run more frequently; however the results show an equal or lower cross-section.

There are few possible explanations for this discrepancy. First, the differing neutron profiles between the experiments make a direct comparison difficult, especially on components that exhibit very low error rates. In addition, the fact that very few events were observed results in a very large margin of error, especially for the crypto engine; it may be that the cross-section values are actually more similar than they appear. Finally, while one may expect that running the test programs more frequently would result in a higher cross-section, it may not be the case. It is possible that the few errors observed in the previous work were due to accumulated upsets in internal hardware registers, and occurred independent of how frequently the test programs were run.

Unfortunately the results from the previous work are quite limited, and make it difficult to demonstrate that the parallel testing methodology presented in this work produces comparable results to a sequential testing methodology.

V. Conclusions

In this paper we have presented a methodology for testing multiple components of an SoC in parallel, and demonstrated this methodology on the Xilinx UltraScale+ MPSoC device. We implemented this methodology at two neutron radiation experiments to test the following components of this device: the A53 cores, the R5 cores, the cryptographic engine, the DMA engines, the OCM, TCM, and CRAM on-chip memories, and the DDR memory controller. We successfully collected error events for all of these components in parallel to maximize the data collection of our limited radiation test resources. The sensitive neutron cross-section of these components was measured at both facilities and shown to have consistent results.

The results demonstrate that such an approach is an effective way to increase the number of errors that can be observed in a given amount of beam time, and can provide a more comprehensive understanding of the error rates of the various components of the SoC. In our future work we will test additional components on this and other SoC devices, investigate the measurement and identification of error propagation between system components, and use the results to predict full system behavior within a complex operating system under radiation.

References

[1] M. Pignol, "COTS-based applications in space avionics," in *Design, Automation & Test in Europe Conference & Exhibition*, Mar. 2010, pp. 1213–1219.

[2] S. Esposito, C. Albanese, M. Alderighi, *et al.*, "COTS-based high-performance computing for space applications," *IEEE Transactions on Nuclear Science*, vol. 62, no. 6, pp. 2687–2694, Dec. 2015.

[3] R. F. Hodson, Y. Chen, J. E. Pandolf, *et al.*, "Recommendations on the use of commercial-off-the-shelf (COTS) electrical, electronic, and electromechanical (EEE) parts for NASA missions - phase II," NESC-RP-19-01490, Dec. 1, 2022.

[4] C. Villalpando, D. Rennels, R. Some, and M. Cabanas-Holmen, "Reliable multicore processors for NASA space missions," in *Aerospace Conference*, Mar. 2011, pp. 2309–2320.

[5] W. Wolf, A. A. Jerraya, and G. Martin, "Multiprocessor system-on-chip (MPSoC) technology," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 10, pp. 1701–1713, Oct. 2008.

[6] H. Quinn, C. Corley, and P. Thelen, "Neutron-induced SEEs in the xilinx versal prime," in *Radiation Effects Data Workshop*, Jul. 2022, pp. 1–8.

[7] W. Stirk, E. Poff, J. Smith, J. Goeders, and M. Wirthlin, "Comparison of neutron radiation testing approaches for a complex SoC," *IEEE Transactions on Nuclear Science*, vol. 70, no. 4, pp. 505–514, Apr. 2023.

[8] D. Hiemstra and A. Baril, "Single event upset characterization of the pentium(r) MMX and celeron(r) microprocessors using proton irradiation," in *Radiation Effects Data Workshop*, Jul. 2000, pp. 39–44.

[9] D. Hienistra, S. Yu, and M. Pop, "Single event upset characterization of the pentium(r) MMX and low power pentium(r) MMX microprocessors using proton irradiation," in *Radiation Effects Data Workshop*, Jul. 2001, pp. 32–37.

[10] D. Hiemstra and A. Baril, "Single event upset characterization of the pentium(r) MMX and pentium(r) II microprocessors using proton irradiation," *IEEE Transactions on Nuclear Science*, vol. 46, no. 6, pp. 1453–1460, Dec. 1999.

[11] G. Swift, F. Fannanesh, S. Guertin, F. Irom, and D. Millward, "Single-event upset in the PowerPC750 microprocessor," *IEEE Transactions on Nuclear Science*, vol. 48, no. 6, pp. 1822–1827, Dec. 2001.

[12] S. E. Michalak, A. J. DuBois, C. B. Storlie, *et al.*, "Neutron beam testing of high performance computing hardware," in *Radiation Effects Data Workshop*, Jul. 2011, pp. 169–176.

[13] D. A. Santos, L. M. Luza, M. Kastriotou, *et al.*, "Characterization of a RISC-v system-on-chip under neutron radiation," in *International Conference on Design & Technology of Integrated Systems in Nanoscale Era*, Jun. 2021, pp. 103–108.

[14] D. A. Santos, A. M. P. Mattos, L. M. Luza, *et al.*, "Neutron irradiation testing and analysis of a fault-tolerant RISC-v system-on-chip," in *International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, Oct. 2022, pp. 31–36.

[15] P. Ramos, V. Vargas, M. Baylac, *et al.*, "Evaluating the SEE sensitivity of a 45 nm SOI multi-core processor due to 14 MeV neutrons," *IEEE Transactions on Nuclear Science*, vol. 63, no. 4, pp. 2193–2200, Aug. 2016.

[16] D. M. Hiemstra and V. Kirischian, "Single event upset characterization of the zynq-7000 ARM® cortex™-a9 processor unit using proton irradiation," in *Radiation Effects Data Workshop*, Jul. 2015, pp. 76–78.

[17] J. D. Anderson, J. C. Leavitt, and M. J. Wirthlin, "Neutron radiation beam results for the xilinx UltraScale+ MPSoC," in *Radiation Effects Data Workshop*, Jul. 2018, pp. 194–200.

[18] S. M. Guertin and M. Cui, "SEE test results for the snapdragon 820," in *Radiation Effects Data Workshop*, Jul. 2017, pp. 155–160.

[19] H. Quinn, T. Fairbanks, J. L. Tripp, G. Duran, and B. Lopez, "Single-event effects in low-cost, low-power microprocessors," in *Radiation Effects Data Workshop*, Jul. 2014, pp. 246–254.

[20] M. Peña-Fernandez, A. Lindoso, L. Entrena, M. Garcia-Valderas, Y. Morilla, and P. Martín-Holgado, "Online error detection through trace infrastructure in ARM microprocessors," *IEEE Transactions on Nuclear Science*, vol. 66, no. 7, pp. 1457–1464, Jul. 2019.

[21] G. S. Rodrigues, F. Rosa, Á. B. de Oliveira, F. L. Kastensmidt, L. Ost, and R. Reis, "Analyzing the impact of fault-tolerance methods in ARM processors under soft errors running linux and parallelization APIs," *IEEE Transactions on Nuclear Science*, vol. 64, no. 8, pp. 2196–2203, Aug. 2017.

[22] B. James, H. Quinn, M. Wirthlin, and J. Goeders, "Applying compiler-automated software fault tolerance to multiple processor platforms," *IEEE Transactions on Nuclear Science*, vol. 67, no. 1, pp. 321–327, Jan. 2020.

[23] B. James, M. Wirthlin, and J. Goeders, "Investigating how software characteristics impact the effectiveness of automated software fault tolerance," *IEEE Transactions on Nuclear Science*, vol. 68, no. 5, pp. 1014–1022, May 2021.

[24] Á. B. de Oliveira, G. S. Rodrigues, F. L. Kastensmidt, *et al.*, "Lockstep dual-core ARM a9: Implementation and resilience analysis under heavy ion-induced soft errors," *IEEE Transactions on Nuclear Science*, vol. 65, no. 8, pp. 1783–1790, Aug. 2018.

[25] M. Bohman, B. James, M. J. Wirthlin, H. Quinn, and J. Goeders, "Microcontroller compiler-assisted software fault tolerance," *IEEE Transactions on Nuclear Science*, vol. 66, no. 1, pp. 223–232, Jan. 2019.

[26] P. M. Aviles, A. Lindoso, J. A. Belloch, M. Garcia-Valderas, Y. Morilla, and L. Entrena, "Radiation testing of a multiprocessor macrosynchronized lockstep architecture with FreeRTOS," *IEEE Transactions on Nuclear Science*, vol. 69, no. 3, pp. 462–469, Mar. 2022.

[27] H. Quinn, Z. Baker, T. Fairbanks, J. L. Tripp, and G. Duran, "Software resilience and the effectiveness of software mitigation in microcontrollers," *IEEE Transactions on Nuclear Science*, vol. 62, no. 6, pp. 2532–2538, Dec. 2015.

[28] A. Lindoso, M. García-Valderas, L. Entrena, Y. Morilla, and P. Martín-Holgado, "Evaluation of the suitability of NEON SIMD microprocessor extensions under proton irradiation," *IEEE Transactions on Nuclear Science*, vol. 65, no. 8, pp. 1835–1842, Aug. 2018.

[29] P. Maillard, M. Hart, J. Barton, J. Arver, and C. Smith, "Neutron, 64 MeV proton & alpha single-event characterization of xilinx 16nm FinFET zynq® UltraScale+™ MPSoC," in *Radiation Effects Data Workshop*, Jul. 2017, pp. 139–143.

[30] P. Maillard, J. Arver, C. Smith, O. Ballan, M. J. Hart, and Y. P. Chen, "Test methodology neutron characterization of xilinx 16nm zynq® UltraScale+™ multi-processor system-on-chip (MPSoC)," in *Radiation Effects Data Workshop*, Jul. 2018, pp. 206–209.

[31] P. Maillard, Y. P. Chen, J. Arver, V. Merugu, A. Shui, and M. Voogel, "Neutron and 64mev proton characterization of xilinx 7nm VersalTM multicore scalar processing system (PS)," in *Radiation Effects Data Workshop*, Jul. 2022, pp. 18–22.

[32] Xilinx, "Device reliability report first half 2021," 2021, p. 93.

[33] D. M. Hiemstra, V. Kirischian, and J. Brelski, "Single event upset characterization of the zynq UltraScale+ MPSoC using proton irradiation," in *Radiation Effects Data Workshop (REDW)*, Jul. 2017, pp. 135–138.

[34] D. S. Lee, M. King, W. Evans, *et al.*, "Single-event characterization of 16 nm FinFET xilinx UltraScale+ devices with heavy ion and neutron irradiation," in *Radiation Effects Data Workshop (REDW)*, Jul. 2018, pp. 275–282.

[35] S. Wender and L. Dominik, "Los alamos high-energy neutron testing handbook," presented at the AeroTech, Mar. 10, 2020, pp. 2020–01–0054.