A Base FPGA Platform for Safer Designs exposed to harsh radioactive environments

Ricardo Zanone
Computer and Electronic Engineering Department
Technological Institute of Aeronautics
São José dos Campos, Brazil

Osamu Saotome
Computer and Electronic Engineering Department
Technological Institute of Aeronautics
São José dos Campos, Brazil

***Abstract*— FPGAs offers a very affordable platform for high-performance computation, while allowing lots of flexibility to perform easy in-field reconfiguration. The easy reconfiguration is achieved by using a memory that holds all the configuration bits allowing for the FPGA internal elements be interconnected to perform custom functions and in most cases this memory is of the SRAM type. A bit flip in this memory could potentially cause a misconnection or changes in the implemented circuit and the final computation could be impacted. This makes the Errors detection and correction in the configuration RAM (CRAM) an important feature of systems exposed to harsh environments. This paper presents a base platform that uses Xilinx's IPs and features of the 7 Series Zynq device to inject, detect, and correct errors while the system allows the dynamic partial reconfiguration (DPR) for run time changes in response to application requirements. The study presented in this paper uses the premise of using a COTS device, without being radiation-hardened, lowering the cost of the implementation, and allowing the deployment of non-safe-critical devices that still can perform under the threats imposed by the surrounding environment. The results found in this paper shows that the memory cache improves the overall performance of the DPR and hardware acceleration more than 300 times in some situations. Automatic Error Injection showed that less than 0.05% of the bit flips caused errors in the logic operation results.**

Keywords*— *SoC, FPGA, COTS, soft error mitigation (SEM), dynamic partial reconfiguration (DPR)

I. INTRODUCTION

The increasing necessity for higher computation performance and flexibility, allowing upgrade and repurposing of the in-field deployed devices, presents a good usage for Systems-on-Chips (SoCs). These devices packs, in the same component, an FPGA, one or more processors, different types of communication buses and many other devices for Input/Output of data.

The most common way of reconfiguring an FPGA is using a flash memory to store a bitstream, generated by the vendor's tool, that contains all the configuration bits that will implement the final logic circuit. During boot, the FPGA SRAM memory is programmed with this bitstream and then the FPGA fabric will start to act as intended by the user's logic implemented. This brings an easy way to reprogram the device but makes the component more susceptible to bit flips (single event upsets). Xilinx claims [4] that their devices are only susceptible to single event effects caused by ionization radiation.

When the radiation hits an SRAM cell it can flip the value of a bit, if the flipped bit is used (essential) in any way by the design, it could alter the functionality of the circuit configured by that cell.

Another possible upset is the latch-up, which can short circuit power rails, and then the device should be power cycled to avoid being damaged or even destroyed.

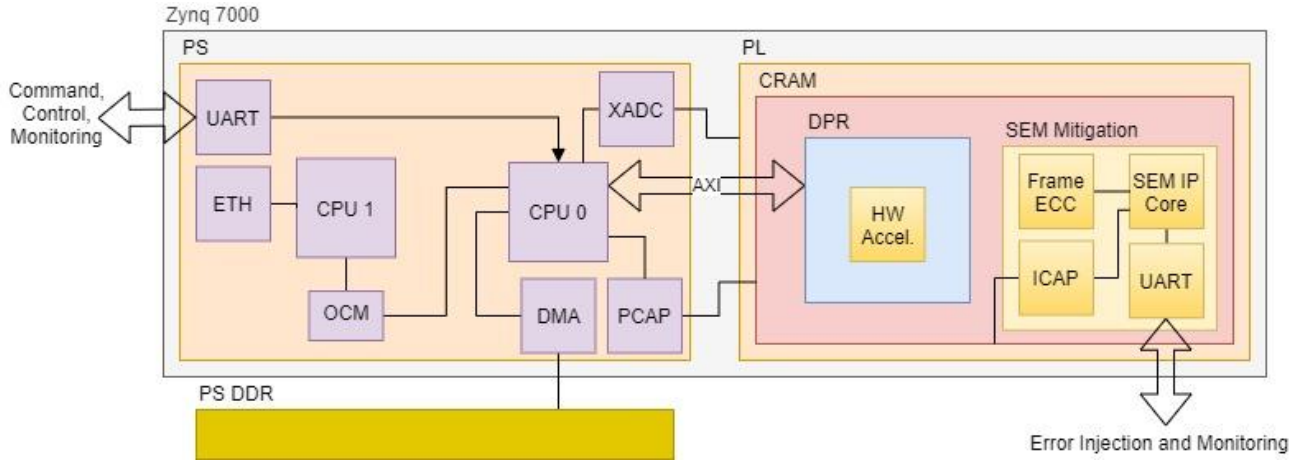
Although the radiation levels are worse in space, even at earth's surface or at higher altitudes (aviation) the devices are susceptible to bit flips caused by radiation, so this kind of error mitigation has more application than only in space programs.

For soft error mitigation, memory scrubbing is one such technic, and extensively studied [8,9]. This paper uses an option provided for free within the Xilinx's tool suite, the SEM IP (Soft Error Mitigation Controller Intellectual Property) Controller [5].

Aside from memory scrubbing, there are other options for error correction, like full or partial reconfiguration [7]. The main downside of total reconfiguration is the time the system will be unavailable due to the reconfiguration, so partial reconfiguration reduces dramatically the downtime because only a small portion of the device/design will change.

Partial reconfiguration has other roles, one of them is to give a new function to the same area of the FPGA, for example, changing a filter type or the arithmetic operation being executed. Another important role of DPR in embedded systems is to upgrade the device while other vital function is still being executed. One example is the reconfiguration of a device that is connected to a computer through a PCIe interface. In a entire FPGA reconfiguration, the PCIe channel will stop communicating with the computer and the computer will need to be rebooted in order to recognize (enumeration process) the PCIe device again, with DPR it is possible to reprogram just the area that is used for some other logic and the PCIe interface continues its normal operation, in this case the PCIe interface can be used to receive this new bitstream an reconfigure the device's memory [6].

Fig. 1. Block Diagram Representation of the proposed platform



A. *Motivation*

Embedded systems are commonly deployed in harsh environment with minimal or no intervention and normally stays with an online operation for a long period between resets, in this case, the effects of the radiation are most critical, because the chances of an upset in an essential bit are higher and the impact in the operation more probable.

Systems that should operate long times between power cycles should have mechanisms for soft error mitigation.

Dynamic partial reconfiguration could be used for error correction and to upgrade functions while others continue its operation without being disturbed.

Most studies present each subject used separated from each other and the ones that implement both, normally implements in a fabric-only device and most of the time in older technologies, without using the ARM processor.

B. *Related Work*

Bruijstens [9] is the only reported work that presents SEM together with DPR in a Zynq 7000 device. Keshk and Asami [10] presents both subjects together but using a fabric-only device.

II. BASE PLATFORM´S ARCHITECTURE

A. *High Level Overview*

The core of the architecture is the Xilinx's Zynq 7000, which is an SoC that has a dual-core ARM A9 and an FPGA fabric in the same die.

The proposed architecture, Figure 1, should be used as a base design for implementing DPR for hardware coprocessing and acceleration while being constantly monitored for soft errors and device's health, such as temperature and voltage.

Those characteristics makes the platform suitable to be deployed in harsh environments, mainly in non-critical instruments or communications [8].

B. *Soft Error Mitigation*

To protect SRAM-based FPGA, Xilinx's SEM IP is being used in this platform. The controller can be accessed from a serial console and provides the status of the configuration memory, injection of errors and operational status.

It is possible to communicate with the controller using the AXI Bus of the processor, but for this study an external UART was attached to the design, providing an easy way to use an external PC with a script to inject error and check how the platform behaves.

C. *Dynamic Partial Reconfiguration*

Dynamic partial reconfiguration is performed through the ARM´s PCAP port.

The full or partial bitstream is loaded to the processor DDR, then using DMA, the file is transferred to the configuration memory of the PL using the PCAP port.

D. *Device´s health monitoring*

Using the XADC present in the Xilinx's 7 series, the temperature and voltages of the device are monitored and can be presented to the operator by a drop-down menu presented on the serial console.

The temperature and voltages being monitored with the XADC are internal values from the SoC.

Since the XADC doesn't measure current consumption, the addition of external sensors should be added to ensure that latch-ups could be detected to improve the reliability of the platform.

III. DPR AND SEM IN THE SAME PROJECT

A. *Configuration Ports*

In Zynq [3], the processor sub-system has control, at power up, of all configurations to the PL using the PCAP port. The system is designed this way mainly because of two factors. The first one is improving the overall security of the design, allowing the ARM processor TrustZone to control the configuration port, only allowing trusted devices access the FPGA ICAP port to configure the device memory, and the second is allowing the FPGA´s configuration at boot time using the ARM's first stage bootloader (FSBL).

The selection of the interface in control of the FPGA configuration is made by a 2:1 mux, controlled by the PCAP_PR register, as in figure 2.

After initial configuration (or at any time), the processor may set the control bit to switch the configuration interface to be accessed internal to the FPGA fabric, then ICAP becomes active.

As in figure 2, the SEM controller uses the ICAP port to perform the CRAM's scrubbing. So, after the initial configuration, the processor should release the PCAP and pass the control to ICAP, permitting the SEM controller monitor, detect and correct errors caused by radiation.

If the processor needs a partial reconfiguration in some area within the fabric, it will need the PCAP control back.

When using the SEM Controller, this action is not as simple as changing the mux [11,12] configuration, the following actions should take place:

1. Put the SEM IP in IDLE mode.
2. Send a SYNC command, stopping the SEM IP frame´s error scanning.
3. Select the PCAP port and perform the dynamic partial reconfiguration.
4. After the DPR operation is done, transfer the control of ICAP back to SEM IP.
5. Perform a soft reset on SEM IP.

Put the SEM IP in OBSERVATION mode.

Figure 2 shows the interfaces that were developed to decouple the SEM IP from the ICAP, switching the control to PCAP perform the partial reconfiguration. The items in yellow permits that steps 2 and 4 are correctly executed.

To stop SEM IP scanning, the controller should be configured to the Idle mode and a SYNC word command, should be sent through the mux to the ICAP. This is accomplished using the output of the "ICAP Sync controller" block that sends the value "0x5599AA66".

When the DPR is finished, the PCAP can switch the control back to SEM controller. This is accomplished commanding the "icap_sel" value and configuring the mux to connect the SEM IP with the ICAP.

After a partial reconfiguration is performed, before the SEM IP return to the observation mode it is necessary to perform a soft reset in the controller, recalculating the FRAME ECC of the CRAM.

If one commands the SEM IP back to the observation mode without the soft reset, the controller will detect the reconfiguration as SEUs and will try to correct them.

If one do not change the SEM IP to the idle mode prior to a DPR, it will continue in observation mode then during partial reconfiguration the changes in the CRAM will be understood as SEUs and the SEM IP will try to correct those bit flips, potentially entering in a loop of error correction and reconfiguration, resulting in a failed reprogramming of the area.

It is important that all PCAP or ICAP operations are completed before switching the control between them.

Figure 2 shows that JTAG has precedence over both ports, so care should be taken when using the JTAG together with PCAP and ICAP, avoiding interference between the ports.

IV. BASIC PARTIAL RECONFIGURATION FLOW

A. *DPR project flow*

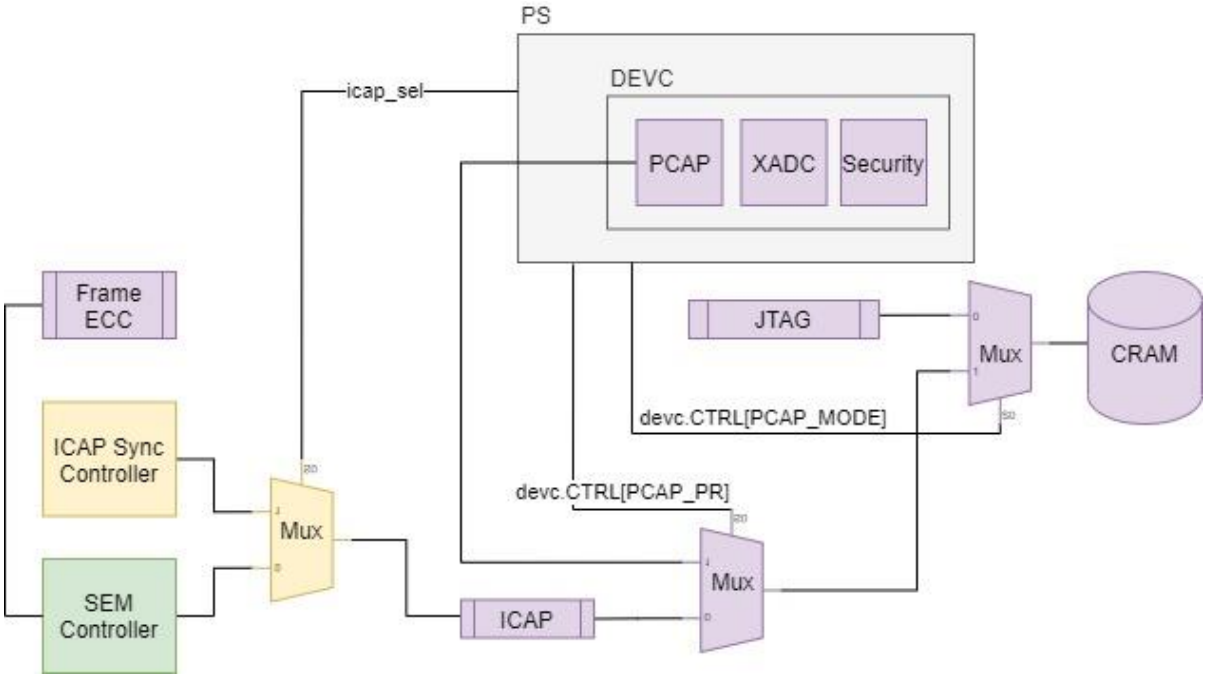
Dynamic Partial Reconfiguration (DPR) flow is supported by Vivado, but not yet fully integrated with the GUI-based project flow [6].

The designs are implemented using the Vivado Tcl based command flow or using a combination of Tcl commands and the GUI.

To facilitate the reproduction of the results and provide a base configuration for other studies using this architecture, a set of scripts were developed to create a project with two PR modules, implementing the following steps:

1) Create the project and block design.
   a) The block design should contain the ARM processing system instance.
   b) The reconfigurable items should have only the interface definitions declared in this step.
2) Synthetize the design, generating all the design checkpoints (DCP) for the static region and blocks of logic.
3) Load the DCP for static and one of the reconfigurable partitions (RP).
4) Define RP properties.
5) Run Design Rule Checker (DRC).
6) Create and implement first configuration.
7) Create all other configurations.
8) Verify all configurations (PR_verify command).
9) Generate bitstreams.

Fig. 2. Configuration ports and paths for the PL's CRAM



V. Hardware Acceleration and High Level Synthesis HLS

Hardware acceleration takes advantage of the reconfigurable device and could be used to offload computations from the processor to hardware's blocks, improving the execution time and freeing the processor to perform other actions and get the results when available.

High Level Synthesis (HLS) flow is supported by Vivado and could be fully addressed using the GUI or Tcl commands.

HLS provides the ability to design the algorithm using C/C++ language and using the tool, translate that for an RTL design in VHDL or Verilog.

This study takes advantage of HLS, implementing a 32x32 Matrix multiplication and the following steps should be performed to create the RTL:

1. Creation of C code and C testbench, describing and testing the accurateness of the design.
2. C functional Verification, using Vivado HLS tool.
3. HLS, using the user constraints.
4. C/RTL CoSimulation.
5. Implementation Evaluation
6. RTL IP export

VI. Automatic Error Injection

Xilinx 7 series SRAM-based FPGA has three main types of memory embedded within the device (1) configuration memory (CRAM), (2) block memory (BRAM) and (3) distributed memory (DM).

The design functionality is loaded in the CRAM, using the bitstream file, which is arranged in frames, protected by error correction code (ECC) and cyclic redundancy check (CRC).

Frames are the smallest addressable segment in the FPGA CRAM, consisting of 101 configuration words of 32 bits each.

Design functionality will change its behavior if one or more essential bits are flipped, characterized as critical bits.

The device's bitstream size is fixed and represents all the configurable hardware present in the device and accessible to the user logic design and includes all the memory configuration (CRAM, BRAM, DM) in its size.

SEM IP perform scrubbing only in CRAM and the essential bits count will be smaller than the bitstream. For error mitigation in the other memories, triple modular redundancy (TMR) voting, ECC or CRC could be applied in the user logic to correct error in BRAM and DM.

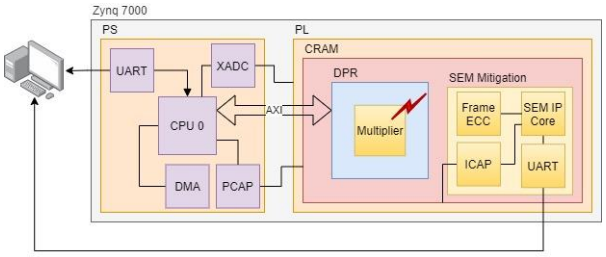
The device used in this study has 25697632 essential bits, but the design will not use all those bits [4], so an automatic test could (1) inject a bit flip in a random address or (2) inject error in all address within a small block of the device. This study uses the second technic.

During the implementation flow of the DPR design, it is constrained the position of the PR in the device, using a primitive of the Vivado toll called "pblock". With the coordinates of this block, Aranda and Maestro [13] demonstrate how to calculate the essential bits address contained in that region.

Using the above method, it was discovered 70182 essential bits in the PR region, so an automatic test could focus on bit flip injection inside that region, checking the design behavior and result.

Figure 3 presents the architecture used, where a Python script sends the bit flip address to SEM IP and check the result of the hardware multiplication against a similar one performed in software, using the ARM A9 core. The script then writes a test report, saving the address and the result of the comparison, where "0" means both results matches or "1" when the hardware result does not match with the software counterpart.

Fig. 3. Automatic error injection architecture



VII. RESULTS

A. Hardware Coprocessing

Multiplication is a commonly used arithmetic computation, present in most embedded devices performing digital signal processing.

It was implemented two hardware coprocessing versions evaluating the acceleration factor against the same calculation in a processor using multiplication in software.

The first implementation used the AXI Lite 4 and a single multiplication operation in hardware. Table I presents the execution time of 1024 multiplications, in SW and HW.

TABLE I. HW VS SW SINGLE MULTIPLICATION

	Time (us)
Hardware	2320
Software	346

It is clear, in this case, that using AXI Lite 4 to perform multiplication is approximated 6.7 times slower than the code executed in software. The main reasons for this are (1) the ARM runs at 650MHz and the Hardware at 100MHz, almost the same factor detected in the time execution and (2) the AXI Lite 4 has a bigger overhead and is not designed for transactions with a lot of data being moved inside the FPGA.

For comparison, using the HLS flow it was designed a second version that performs a 32x32 matrix multiplication algorithm, using the AXI Stream interface.

The AXI Stream is designed for moving high volume of data with minimal overhead in its protocol. The design implemented uses DMA to move data in and out of the processor. Table II presents the execution time of an 32x32 matrix multiplications, in SW and HW.

TABLE II. HW VS SW 32X32 MATRIX MULTIPLICATION

	Time (us)
Hardware, cache enabled	53
Software, cache enabled	1567
Hardware, cache disabled	69
Software, cache disabled	24099

It is clear, in this case, that using AXI Stream improves the computation and achieve the hardware acceleration pretended, offloading the ARM processor. When using the DDR's memory cache, the improvement of the hardware coprocessing is approximately 30 times and 350 times with disabled caches.

B. Automatic Error Injection

In VI, it was presented that the number of injections was dramatically reduced, focusing in the area where the multiplication hardware is implemented. Table III presents the results from the performed injections.

TABLE III. INJECTION STATISTICS

Total of Frames with essential bits in the PR	70182
Total of Injections executed	21935
Erroneous multiplication result	7
Injection with AXI Bus locked	126

From table III, the frames that contain essential bits inside of the PR region is only 3.66% of the total essential's bits of the device.

Table III shows that it was injected approximately 30% of the addresses with essential bits, this happened because after this, the injected errors started to hang up the processor when the AXI bus was accessed. Because of this behavior, the script was finished, as it is not conclusive when the AXI Bus hangs, the result is not show, and it is considered a critical failure.

From those injection, only 7 addresses presented an erroneous multiplication result, and this was caught comparing the result of the HW multiplication against the same operation coded in software.

Figure 4 shows an excerpt of the automatic test report generation, presenting the address of the bit flip and the result of the device under test (DUT), in this case the multiplier, were a "0" means that HW and SW results are equal, Figure 5, and a "1" indicates that the HW result is different from the SW, Figure 6.

C. Dynamic Partial Reconfiguration

TABLE IV. PARTIAL RECONFIGURATION TIME

	Time (us)	Throughput (Mb/s)	Cache
First DPR	897	143	Enabled
All others PR	1.35	95195	Enabled
First DPR	1186	108	Enabled and invalidated
All others PR	576	222	Enabled and invalidated
First DPR	984	130	Disabled
All others PR	6	21117	Disabled

Sultana [7] mentions that PCAP theoretical throughput is 400Mb/s, using the 32 bits platform, achieving an average of 140Mb/s, that is not far from the results obtained in this study.

Table IV shows that when the first PR is performed, the throughput is closer to the theoretical and from the second and beyond, within some configurations, the speed is unreal. This was narrowed down as a side effect of the small bitstream used in the PR and the Zynq´s DDR cache. The cache, when enabled, presents almost 96Gb/s of write speed, but when the cache is enabled and commanded to invalidate its contents before each PR, the speed goes closer to the average and lower than the theoretical value.

This is due to the PCAP construction that could only operate using DMA from the Zynq DDR and so it will use the cache if it is not explicity disabled or invalidated.

Achieving faster reconfiguration could be obtained using custom controllers, as Vipin [14] presented using the Zynq device.

Fig. 4. Excerpt from Test Report output.

```
Frame: C00035E329 | DUT Checker: 0
Frame: C00035E32A | DUT Checker: 0
Frame: C00035E32B | DUT Checker: 0
Frame: C00035E32C | DUT Checker: 1
Frame: C00035E32D | DUT Checker: 0
```

Fig. 5. . HW and SW multiplier result are equals

```
> 4
First operand: 9
Second operand: 9
Total Ticks: 756088
Total Time: 2326 us
HW Result: 81
Total Ticks: 112648
Total Time: 346 us
SW Result: 81
```

Fig. 6. Wrong HW result, due to a single event upset (bit flip)

```
> 4
First operand: 9
Second operand: 9
Total Ticks: 757094
Total Time: 2329 us
HW Result: 10485849
Total Ticks: 112589
Total Time: 346 us
SW Result: 81
```

D. Future Work

Automatic test showed a catastrophic flaw in the AXI bus, where a hang up was detected and it was only recovered after a reset and a full reconfiguration and this should be addressed in a new study, improving the AXI bus to avoid this situation and implementing the automatic detection of an eventual lock to reset the processor and reconfigure the FPGA.

The SEM IP has a feature that classifies the bit and then provide an interface to inform that an error occurred in this essential bit or not. This feature should be implemented to improve the reliability of the design, allowing the design to be aware that the controller detected an error in a critical bit and then inform the processor so that every computation executed during this upset should be reevaluated to avoid any potential error.

If using partial reconfiguration for error correction in the frames, the classification status will improve the timing schedule, just reconfiguring when a critical bit is flipped and classified as essential/critical.

As seen during practical tests and Bruijstens [9], the use of the enhanced repair mode of the SEM Controller together with the use of DPR is not optimal, as the time taken by the controller to fully initialize, using the XC7Z020 device, is 2.9s [5]. A new study using the repair mode is a good improvement to the final solution.

At this moment, only SoC's temperature and voltage are monitored using the FPGA internal sensors, but a beneficial addition is an external sensor to measure the current consumption, providing a mean for accurate latch-up detection and power consumption calculation.

Use of the second core of the processor, adding the support for Linux and ethernet, facilitating external communication and control.

CONCLUSION

The increase in high performance computation usage in embedded system makes the FPGA an alternative that consumes relative low power and can be easily updated and reprogrammed.

Radiation hardened devices are normally expensive and imposes some restrictions in the end user and acquisitions, so the use of COTS SRAM based FPGA is interesting in non-critical devices, using the error mitigation characteristics presented throughout this study.

It was showed that dynamic partial reconfiguration can be used together with soft error mitigation using memory scrubbing by following some simple steps before and after each event.

An automatic error injection platform was presented, with the support of bit flip injection, automatically checking the DUT performance with the production of a test reports and results statistics, showing that time could be saved with the focused injection in specific areas.

The automatic error injection is a cost optimized solution for extensive testing, checking that the mitigations are effective before a certification could be applied to the final product.

If one fails to implement an effective error mitigation and correction, the design could suffer from a critical failure causing unexpected behavior and a risk to its use.

REFERENCES

[1] A. Guerrieri, S. Kashani-Akhavan, P. Lombardi, B. Belhadj and P. Ienne, "A Dynamically Reconfigurable Platform for High-Performance and Low-Power On-Board Processing," 2018 NASA/ESA Conference on Adaptive Hardware and Systems (AHS), 2018, pp. 74-81, doi: 10.1109/AHS.2018.8541452.

[2] A. Guerrieri, S. Kashani-Akhavan, M. Asiatici and P. Ienne, "Snap-On User-Space Manager for Dynamically Reconfigurable System-on-Chips," in IEEE Access, vol. 7, pp. 103938-103947, 2019, doi: 10.1109/ACCESS.2019.2931475.I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[3] Xilinx, "Zynq-7000 SoC Data Sheet: Overview (DS190)," 2018.

[4] Xilinx, "A Practical Look at SEU, Effects and Mitigation", 2016.

[5] Xilinx, "LogiCORE IP Soft Error Mitigation Controller v4.1", 2018.

[6] Vipin, Kizheppatt. "FPGA Dynamic and Partial Reconfiguration : A Survey of Architectures , Methods , and Applications." 2018.

[7] B. Sultana, A. Ullah, A. A. Malik, A. Zahir, P. Reviriego, F. B. Muslim, N. Ullah, and W. Ahmad, "VR-ZYCAP: A Versatile Resourse-Level ICAP Controller for ZYNQ SOC," Electronics, vol. 10, no. 8, p. 899, Apr. 2021.

[8] F. Brosser, E. Milh, V. Geijer and P. Larsson-Edefors, "Assessing scrubbing techniques for Xilinx SRAM-based FPGAs in space applications," 2014 International Conference on Field-Programmable Technology (FPT), 2014, pp. 296-299, doi: 10.1109/FPT.2014.7082803.

[9] Bruijstens, D.P., "Reliability of SRAM-based FPGAs", 2018

[10] Keshk, Mohamed El-Hady & Asami, Kenichi, "Fault Injection In Dynamic Partial Reconfiguration Design Based On Essential Bits", 2018

[11] Xilinx, "Zynq 7000 - Switching between ICAP and PCAP Recommendations", 2016.

[12] John Ayer Jr, "Dual Use of ICAP with SEM Controller", XAPP517 Ver. 1.0, Dec., 2011.

[13] L. A. Aranda, A. Sánchez-Macián and J. A. Maestro, "ACME: A Tool to Improve Configuration Memory Fault Injection in SRAM-Based FPGAs," in IEEE Access, vol. 7, pp. 128153-128161, 2019, doi: 10.1109/ACCESS.2019.2939858.

[14] K. Vipin and S. A. Fahmy, "ZyCAP: Efficient Partial Reconfiguration Management on the Xilinx Zynq," in IEEE Embedded Systems Letters, vol. 6, no. 3, pp. 41-44, Sept. 2014, doi: 10.1109/LES.2014.2314390.