On Enhancing Fault Injection's Capabilities and Performances for Safety Critical Systems

On enhancing fault injection's capabilities and performances for safety critical systems

Stefano Di Carlo, Giulio Gambardella, Paolo Prinetto
Politecnico di Torino
Dipartimento di Automatica e Informatica
Corso Duca degli Abruzzi 24, I-10129, Torino, Italy
Email: {name.familyname}@polito.it

Frank Reichenbach, Trond Løkstad, Gulzaib Rafiq
ABB Corporate Research, Norway
Wireless and Embedded Systems group
Bergerveien 12, 1396, Billingstad, Norway
Email: {name.familyname}@no.abb.com

***Abstract*— The increasing need for high-performance dependable systems with and the ongoing strong cost pressure leads to the adoption of commercial off-the-shelf devices, even for safety critical applications. Ad hoc techniques must be studied and implemented to develop robust systems and to validate the design against all safety requirements. Nonetheless, white-box fault injection relies on the deep knowledge of the system hardware architecture and it is seldom available to the designer. Furthermore it would require enormous simulation time to be carried out. This work presents an enhanced architecture for fast fault injection to be used for design-time coverage evaluation and runtime testing. A test case will be presented on Xilinx Zynq system on programmable chip, suitable for design-time diagnostic coverage evaluation and online testing for safety-critical systems resorting to the proposed fault injection methodology.**

Keywords*— dependability; fault injection; safety; testing; FPGA; diagnosis; functional safety; system-on-programmable-chip.

I. Introduction

Moore's law [1] has served as goal and motivation for consumer electronics manufacturers in the last decades. The results in terms of processing power increase in the consumer electronics devices have been mainly achieved due to cost reduction and technology shrinking. However, reducing physical geometries mainly affects the electronic devices' dependability, making them more sensitive to soft-errors [2] like Single Event Transient (SET) of Single Event Upset (SEU) [3] and hard (permanent) faults, e.g. due to aging effects like electromigration [4].

Accordingly, safety critical systems often rely on the adoption of old technology nodes, even if they introduce longer design time w.r.t. consumer electronics. In fact, functional safety requirements, due to international directives like IEC61508 [5], are increasingly pushing industry in developing innovative methodologies to design high-dependable systems with the required diagnostic coverage.

More recently, commercial off-the-shelf (COTS) devices adoption began to be considered for safety-related systems. Real-time requirements, the need for the implementation of computationally hungry algorithms and lower design costs are pushing industries in this direction. These design choices have a significant impact on the dependability capabilities and diagnostic coverage measurements. Safety-related system designers will mainly adopt COTS devices as black-box modules, facing the problem of testing, diagnosing and coverage evaluation from a different perspective w.r.t. the provider one.

Former approaches for dependability evaluation were usually based on fault injection campaigns [6]. Fault injection is commonly intended as a dependability validation and evaluation technique for fault tolerant systems. It consists of several operations to be performed at design time to assess the design fault tolerance. Such fault injection campaigns can be either performed in hardware (e.g., modifying the value of input pins [7-8] or heavy ion beam radiation [9]), in software (e.g., modification of memory data [10-11]), simulated (e.g., simulating the RTL description and altering signal values [12]) or emulated (e.g., relying on FPGA prototypes [13-15]). Usually, the FARM model [16] is indeed used to characterize the fault injection input and output domains, in which a set of *Faults* and *Activations* in corresponds to a set of *Readouts* and a set of derived *Measures*.

Fault injection can further be used for design-time validation and diagnostic coverage evaluation to enhance testing capabilities. Safety related systems adopt various dependability enhancement techniques, like error correction code (ECC), to reach the needed safety integrity level (SIL). Moreover, such modules should be tested online periodically to monitor their effectiveness. Online error injection can provide a solution to deal with such testing needs.

This paper presents an enhanced architecture able to provide fast fault injection capabilities for safety critical embedded systems. The overall idea investigated here is to inject transient faults into the memory (e.g., the executable code or the data area) of an application by means of an external hardware manipulator, to allow almost concurrent fault insertion and program execution. Such architecture could be used to fast perform fault injection campaigns, like flipping all bits in the instruction and data memory to analyze their effects and severity on a processing system. Furthermore, the presented architecture can be exploited to facilitate online ECC engine testing by allowing runtime memory error injection. Two possible cases of study will be presented resorting to their implementation on a Xilinx Zynq [17] device.

The paper is organized as follows: Section II gives an overview on fault injection, listing and comparing the different approaches. Section III will present the proposed fast fault injection architecture, which implementation details and possible case of study will be highlighted in Section IV. Section V will present the experimental setup and the achieved results, while Section VI will draw the conclusions.

II. Fault Injection background

The emergence of high dependable electronic systems for safety critical applications has led to the need for fast, reliable and affordable methodologies to assess their actual reliability. Fault injection campaigns are usually used in this framework to have precise diagnostic coverage data. Nonetheless, during the design process, usually several trials and errors are made to reach the final solution. Thus, in the time-to-market driven era, fault injection has to be speeded up according to develop products with minimum time-to-market. More recently, the increasing need for high diagnostic coverage in safety critical systems, fault and error injection has been used to increase test effectiveness. In fact, error injection at runtime can be exploited to either increase test coverage and/or decrease testing time (e.g., allowing easy and fast ECC module testing).

Fault injection techniques can be classified according to several objectives [18][19]. Hereinafter, the possible techniques will be tailored to the target fault injected module (microprocessor-based designs are considered), depending whether the RTL description is available (i.e., *white-box*) or not (i.e., *black-box*). For both cases, some possible approaches will be presented highlighting their pros and cons. It should be noticed that black-box approaches can be used in the white-box case (availability of RTL description) as well.

A. White-box approaches

Whenever the designer has the complete view of the internal module, several approaches can be selected to perform fault injection. Depending on the requirements and needs, the user can adopt:

a) Simulation-based fault injection: it is performed by simulating the design injecting faults by altering the internal signals. It features very high design coverage evaluation, thanks to the high accessability and controllability, but it requires very high simulation time. The lower the simulated level (e.g., gate or RTL) the higher is the accuracy, thanks to the fine-grained fault injection, and the simulation execution time.

b) Emulation-based fault injection: it is performed by emulating the design on a configurable platform, usually FPGA, injecting faults on the FPGA-implemented design. The actual fault injection can be either performed: (i) by reconfiguring the device with a fault-injected configuration file or (ii) by instrumenting the design adding extra logic that manages the fault injection directly on hardware. The former approach experiences high time overhead, since total or partial reconfigurations are required for each injected fault, the latter suffers the need of ad-hoc hardware module to be designed and implemented, modifying the original design, and area overhead.

B. Black-box approaches

In order to perform fault injection on black-box hardware modules, special practices have been designed to overcome the restricted accessibility without incurring in high accuracy lost. These approaches can be usually applied in COTS as well, since they do not rely on the availability of internal access or view.

a) Simulation-based fault injection: it is performed by simulating the system, relying on its behavioural model (if available). [20] presents fault injection results achieved relying on QEMU models. They feature lower execution time w.r.t. white-box simulation-based approaches, since the model used is at a higher level of abstraction, but they loose in terms of accuracy. This kind of approach can be useful to identify error propagation in the system but it does not give any detailed information on the diagnostic coverage.

b) Software-implemented fault injection: it consists of injecting faults by software manipulations, that can address both data and instructions to be executed on the target device [21]. The presence of faults in the hardware is reproduced at software level,. Faults can be injected either at compile time or at runtime. These kind of techniques can perform at-speed, guaranteeing fast process at different levels in the software stack (from drivers up to operative systems) but they feature limited accessiblity, since faults can be injected on the software addressable part, only.

c) Hardware-implemented fault injection: it is the more intrusive techniques since it is carried out on the final hardware design. Such a hardware injection can be performed resorting to two different main approaches: (i) perturbing the environment parameter, using heavy ion beam radiations or by placing the device in an electromagnatic field (also referred as injection without contact); (ii) accessing the input pins of the device, modifying the value of signals, exploiting debug infrastructures like scan chains or by mean of active probes (also referred as injection with contact). The former approach feature at-speed performances, at the cost of lower observability. In fact, it is not known if and where the fault has been injected, but it relies on probabilistic evaluation. The latter approach efficiency is contingent with the level of accessibility. As an example, if the debug scan chain is exploited, the longer is the chain the higher is the accessiblity, since more parts of the system can be excited, but requiring more time to be scanned.

This work proposes a methodology that overcomes limitations of previous approaches by combining the benefits of both software and hardware implemented fault injection, assuming a black-box approach.

The goal is to provide a methodology to inject faults in the system by manipulating the instructions to be executed by the system-on-chip and check its behaviour depending on the injected fault, thus providing design time fault severity analysis.

The same approach can be used to dynamically inject faults, at runtime, enhancing memory accessibility for testing purposes.

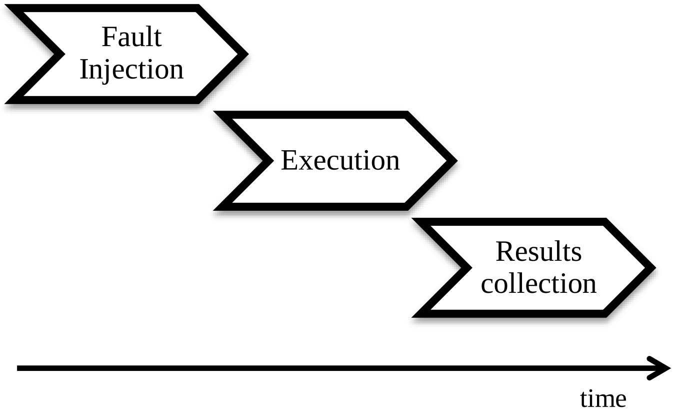


Fig. 1. Standard design-time fault injection process

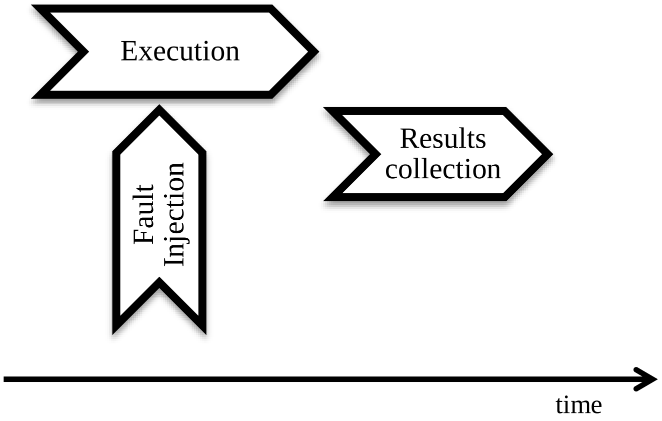


Fig. 2. Proposed design-time fast fault injection architecture

The next section will analyze the overall idea of the proposed methodology to fasten the fault injection process, highlighting the main pros and cons about its effective implementation.

III. Fast Fault Injection architecture

Fault injection is commonly performed offline, at design time and often is part of a campaign, in which a set of faults are injected subsequently. As shown in Fig. 1, three main phases can be identified, and should be repeated for each injected fault.

1. *Fault Injection*: the actual fault is injected in the device, resorting to the chosen fault injection methodology;
2. *Execution*: the program or the simulation is executed on the injected platform;
3. *Results collection:* the results of the execution on the injected platform are gathered and collected, for following analyses.

Either online or offline, the collected results are then compared with the golden data to provide the diagnostic coverage analysis. This kind of modular approach allows the designer to easily automate the fault injection campaign, for example resorting to scripting, but incurs in high execution time due to the pipelined execution. The main idea, as shown in Fig.2, is to parallelize the tasks execution in order to decrease the overall execution time.

In fact, fault injection tasks can be performed during the execution. Thanks to this, the overall fault injection campaign time will decrease, especially considering the big number of faults being injected. Nonetheless, depending on the adopted fault injection methodology, such a parallel execution can be either impossible or its adoption results in poor gain.

Let us consider hardware implemented fault injection using on-chip debug (OCD) infrastructures [8][22][23]: in this case, in order to inject faults, the system must move to debug mode to inject the fault in the memory element, and then switch again in normal mode to execute the program. Check-pointing [24] is a common practice in this field to improve timing performances. The execution is carried out until a checkpoint, the system state is stored and the fault injected. In this case, the next injection will not require the execution from the beginning.

The fast fault injection implementation is instead straightforward for simulation-based approaches, by using force/release approach in HDL simulation [25]. Thanks to this feature available in modern HDL simulators, the simulation is speed up and faults are inserted concurrently to the program execution. Nonetheless, the achieved performances are far from being comparable to hardware-implementations.

Another case of online fault injection is the instrumentation-based emulation fault injection. The instrumentation is often based on multiplexing the original signal with a wired connection to logic 0 or logic 1 [26], to inject stuck-at faults, or xor-ing registers content by mean of external signal in order to emulate soft errors [27]. Clearly, by acting on the signals with the proper timing, the fault can be injected during the execution quite easily.

The implementation of the here proposed fast fault injection architecture is based on the chance of modifying the instruction and data memories of the processor by means of a hardware implemented fault injector, external to the processing system.

The injection process is performed by the hardware module accessing the memory during the execution phase, and modifying it at runtime and writing back the faulty word before it is accessed by the processing unit. Some synchronization is required between the processor and the injector to avoid concurrent accesses to the same memory locations. This prevents uncertainty in what word (i.e., the injected or the fault-free one) is actually accessed by the processing platform and executed.

The module can be implemented either to manage fault injection campaigns, allowing the fault injection in the whole memory space subsequently, or to flip specific memory location by setting its configuration registers. In fact, the presented fault injector, thanks to its small footprint and area occupation, can be further exploited. In fact, by virtue of its flexible and configurable architecture, it can be used to inject errors in the memory at runtime as well (see Fig. 3).

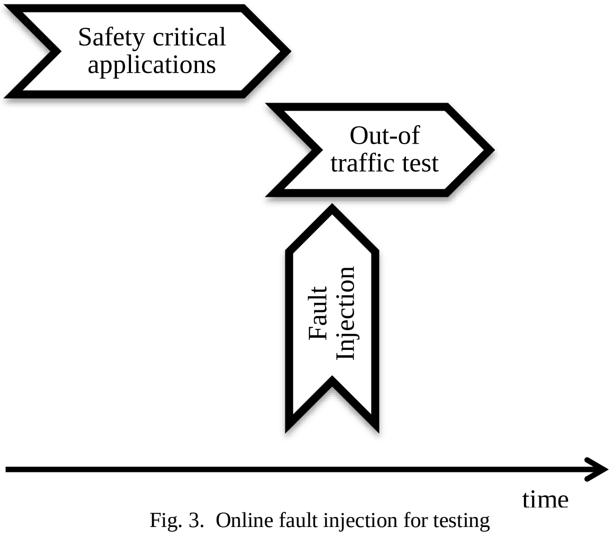


Fig. 3. Online fault injection for testing

The memory location to be flipped can be chosen by setting the fault injector configuration registers (e.g., address and bit position). Designers can select registers values, and decide when to inject the fault in the chosen location. This on-the-fly fault injection increases the system controllability, allowing the injection to be performed in the field, possibly shorten online and out-of-traffic testing time.

In the next section two different cases of study will be presented, adopting the fast fault injection methodology in different scenarios. Both are based on their implementation on the Xilinx Zynq [17], allowing both design-time fault injection (i.e., managing the whole campaign) or online error injection for testing purposes. A comparative study will be presented in order to show when the designer should opt for the first or the second solution, depending on the specific requirements.

IV. Cases of Study

Systems on Programmable Chip (SoPC) are platforms combining the high-efficiency of ASIC-implemented processors, in terms of power and performances, with the intrinsic flexibility of programmable logic, usually SRAM-based Field Programmable Gate Array (FPGA), in a single die [28-30].

The idea of exploiting SoPC features to fasten fault injection is not new. Several works have been presented, mainly based on connecting on-chip debugging to the programmable logic to speed up the fault injection [22][31][32]. Even if they provide good observability, the time required to scan the whole registers chain is relatively high, thus incurring in latencies due to intrinsic JTAG nature. Conversely, the cases of study presented here exploit standard connections among the main modules composing SoPC, to provide high speed and configurable fault injection capabilities.

The target system is intended to be composed of:

- *Processing System*: the processing part of the SoPC, embedding CPU (in modern SoPCs, usually ARM-based), caches and some peripherals;
- *Programmable Logic*: is the configurable part of the system, usually SRAM-based FPGA, in which we implement the hardware fault injector;
- *Memory*: the locations storing data and instructions. They can reside either in an external memory (e.g., DDR), inside the device (e.g., on-chip memory) or in the programmable logic (e.g., Block RAMs)

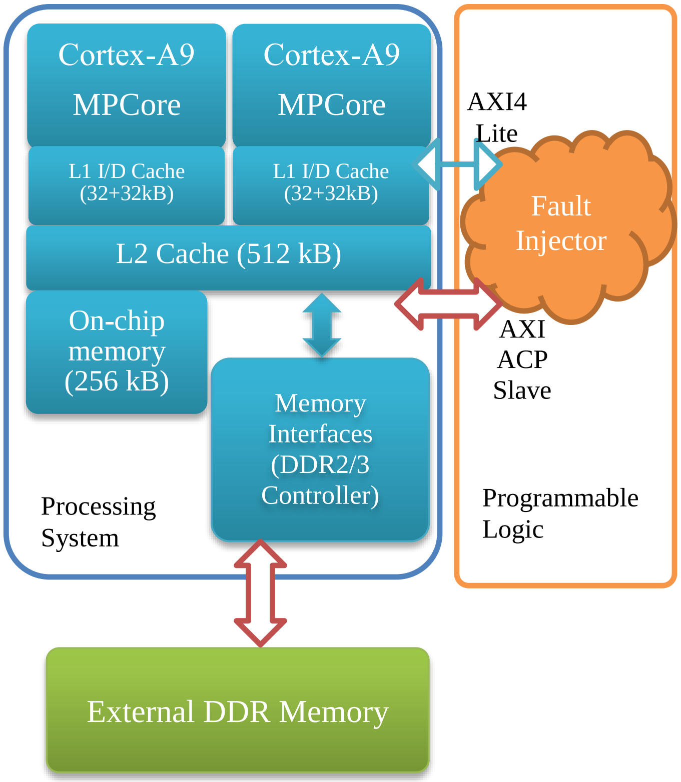
The two presented scenarios will resort to different memory subsystems. In the former one, external DDR stores program and data. The latter one uses RAM blocks available in the programmable logic.

For the sake of brevity, both scenarios will be presented based on their implementation on Xilinx Zynq platform, only, but the same concepts can be easily adapted for other SoPC platforms.

A. Scenario 1 – External DDR

The program under execution is stored in an external DDR memory and directly accessed by the processing system through the memory interfaces block, embedding the DDR controller (see Fig. 4), directly connected to the physical memory.

The fault injector module is implemented in the programmable logic. It addresses the memory location to be read and modified through the AXI ACP (Accelerator Coherency Port) slave, in order to avoid incoherency between caches and central memory and incurring in unpredictable executions. In



case caches are disabled (for timing predictability reasons) AXI HP (High Performances) can be used instead.

Furthermore, the AXI4-Lite bus connection can be used to address the configuration registers of the fault injector module, providing some synchronization between fault injector and processing system. These registers are needed especially to provide online fault injection capabilities, since they allow the user to inject faults in different locations at different time. This is commonly used in out-of-traffic testing in order to achieve higher diagnostic coverage. The synchronization signals (e.g., Start, Done) are needed to trigger the injection process and to notify its proper execution to the processing system.

B. Scenario 2 – Internal BRAM

The program under execution is stored in Block RAMs (BRAM in Fig. 5) in the programmable logic, accessed through the AXI bus (programmable logic to memory interconnect). The BRAM is implemented in the programmable logic as a dual port RAM [33]. Thanks to this intrinsic dual port nature, it allows access to both processing system and fault injector without requiring access sharing or multiplexing.

The fault injector module is hardware-implemented in the programmable logic. It is connected to the BRAM second access port, while the first port is accessed by the Processing system through the Programmable Logic to Memory Interconnect module. This module allows high performances communication between programmable logic and processing systems, exploiting AXI HP ports.

As in Scenario 1, the fault injector module is connected via AXI Lite bus to the processing system to both exchange synchronization signals and configurations.

C. Scenarios Comparison

The two presented scenarios differ mainly on the memory architecture, while they share the same overall fault injector module. In both cases the programmable logic hosts the fault injector module, connected through AXI4 Lite to the processing system. In scenario 1, AXI ACP (or AXI HP with cache disabled) slave is used to communicate with the external DDR memory, since it is not possible to directly address DDR from the programmable logic. Conversely, scenario 2 uses the programmable logic to memory interconnect module to deal with the communication between BRAMs and Processing System.

In terms of timing performances, DDR memory is faster than BRAMs, especially when accessed in burst mode by the Processing System. Furthermore, external DDR memory can be as big as needed, so there are almost no limitations on the code dimension. On the contrary, there are strong limitations on the BRAMs, since those are quite limited resources in the programmable logic, depending on the Zynq device, for a maximum of 3,020 kB.

The main advantage of BRAMs adoption is their dual port nature. This feature allows a very fast and easy way to access the memory content and to modify it, without incurring in latencies due to bus accesses and connections. Thanks to this, the scenario adopting BRAMs as main memory fits perfectly with the concept of fast fault injection presented in Section III.

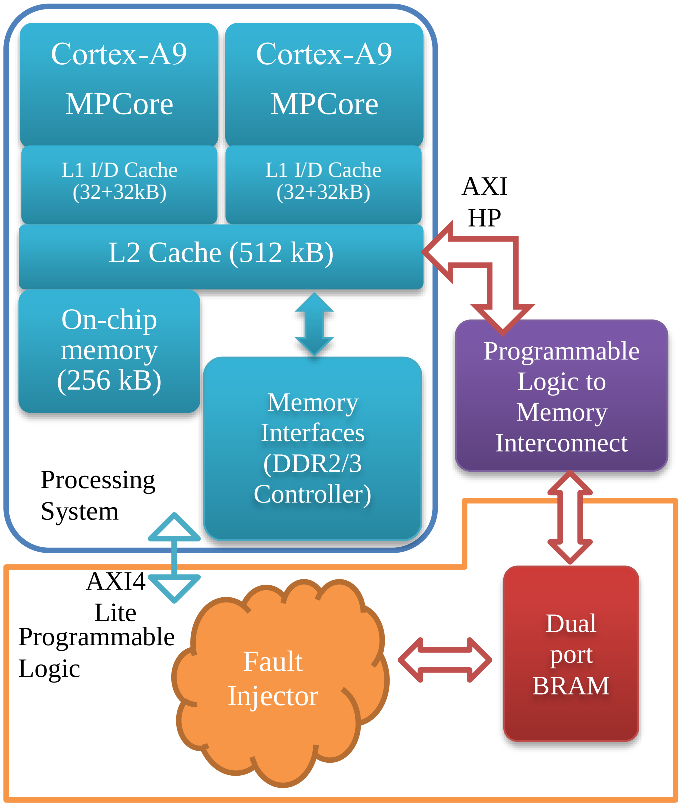


Fig. 5. Scenario 2 – block diagram

When the system is being reset and the program execution starts, in parallel the fault injector can read, modify and write back the injected word in the memory by addressing the port of the BRAM in *WRITE_FIRST* mode.

Differently, DDR memory accesses (injection process and program execution) should be performed in separated time slots, requiring further synchronization between the Processing System and the fault injector, thus slowing down the whole fault injection process. This is due to the intrinsic single-port nature of the DDR memory, that can be accessed by both systems relying on the same interface (i.e., hardwired in the Processing System), only. Furthermore, the fault injector implemented on the programmable logic can access the DDR controller as a slave on the ACP bus, thus influencing the cache that will be flushed and updated (so losing the performances gain due to caching).

According to the specific features of both the scenarios, it should be noticed that scenario 2 should be preferred whenever the memory space provided by BRAMs is enough, since the fault injection process is faster and easier, since low sync is required. This is true either in case of online fault injection and design-time, allowing fast fault injection. Whenever the required space for instructions and data does not fit with BRAMs, external DDR memory provides the required capability. The fault injection process is slower, due to the required execution stall to read, modify and write back the injected fault.

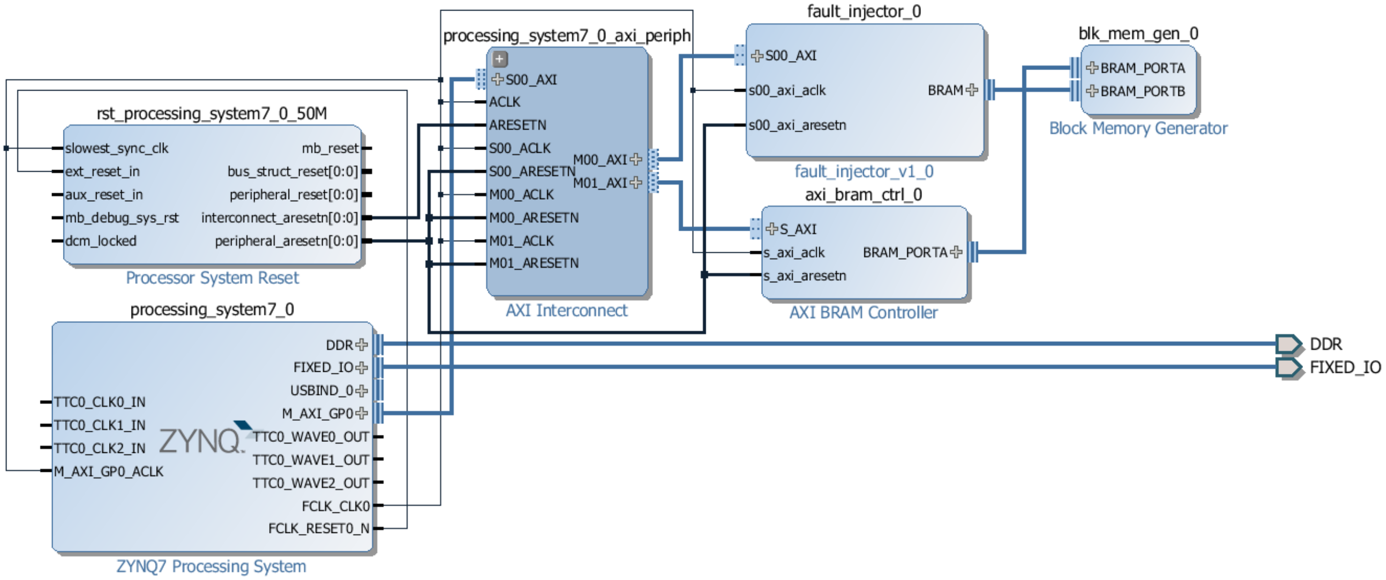


Fig. 7. Fault Injector IP integration in Vivado for scenario 2

V. EXPERIMENTAL RESULTS

The experiments were conducted on a Zedboard (Zynq Evaluation and Development board) [34], with a Zynq XCZ7020-1CLG484C SoPC, including two Micron DDR3 128 Mb x 16 memory components, with a 32-bit interface, for a total of 512 MB, working at 533 MHz, without ECC support. The number of available Block-RAMs in the device is 140, for a total of 560 kB. BRAMs can be configured with parity or ECC hamming codes. The design process has been made using Vivado Design Suite 2013.4 [35]. During these experiments, both instruction and data caches of the processing system were disabled, in order to increase the predictability of the execution. In this case, in scenario 1, either AXI HP or AXI ACP can be used to connect the fault injector to the DDR controller, since cache coherency (e.g., achieved by adopting AXI ACP) is not needed.

The fault injector module has been hardware implemented in the programmable logic as an IP core connected via AXI-4 Lite. Three separate parameters must be provided to the IP, by acting on its configuration registers: *base_address* from which starts the injection, *word_number* in which inject faults sequentially, *starting_bit* identifying which bit should be flipped and *ending_bit* identifying the last bit to be flipped.

The FSM shown in Fig. 6 list the operations performed by the injector module. It basically consists of two nested loops scanning the addresses in which inject the faults, and the bit in each word. A temporary register is used to store the value read from the memory, in order to avoid reading the same value when the fault is injected in different bits of the same word.

When the module is used for single online fault injection, setting *starting_bit=ending_bit* and *word_number=1*, the module will flip a single bit at the desired address (*base_address*). The fault injector hardware module, implemented in the Zynq programmable logic, works up to a maximum frequency of *167 MHz* and requires 268 slices, taking into account the implementation of the FSM, the configuration registers and the AXI4 Lite bus connection.

Scenario 1 has been implemented using AXI HP port. The whole injection process of a single 32 bit word accessed from the DDR, manipulated and written back took in average *463 ns*, and requires the stall of the program execution during that period. Since the execution is interrupted, the possible DDR burst accesses are stopped as well, incurring in further timing penalty. In the worst case, the timing penalty will be of *206 ns*.

Scenario 2 has been implemented (as shown in Fig.7) resorting to a direct connection among BRAMs (*blk_mem_gen* in Fig.7) and fault injector. BRAM addresses are non-cached as DDR in scenario 1. The whole injection process in this case requires *87 ns* in average (working at 150 MHz both BRAMs and fault injector) and does not require any stop in the execution.

Previous approaches relied on the adoption of host platforms to setup the fault injection, by modifying the software image on the host platform before downloading for execution. In this case, for every injected fault, the modified source code should be

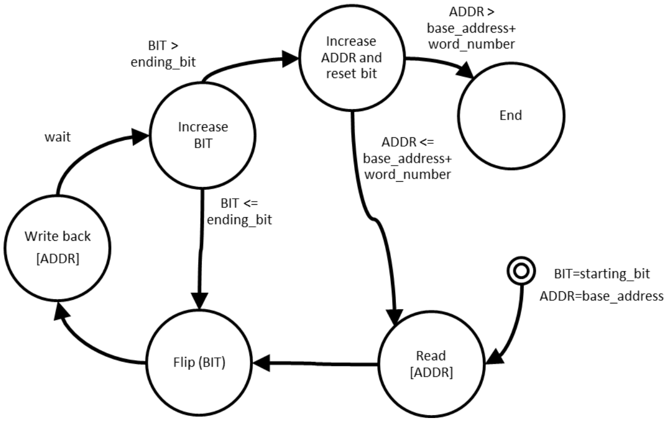


Fig. 6. FSM of the fault injector module

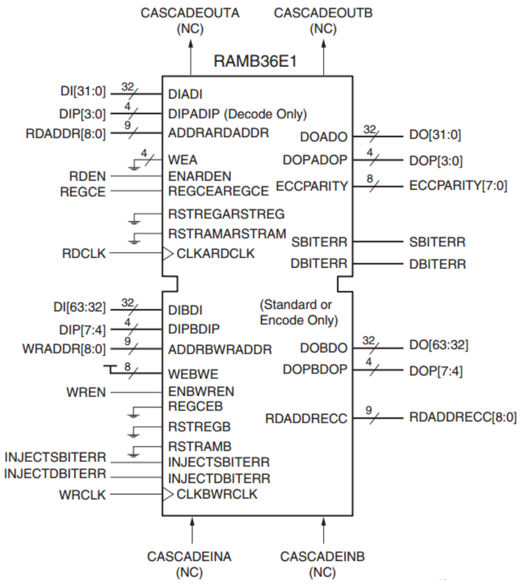


Fig. 8. Xilinx Block-RAM with ECC and error injection [33]

downloaded to the platform before launching the execution. In the Zedboard, this programming process requires the information download through the JTAG chain. For a simple bare-metal hello world application [36], it requires *1.2 s*. For more complex applications, the time required to download the code increases proportionally with the dimension. Clearly, both scenarios 1 and 2 outperform of several orders of magnitude previous approaches.

Special case has been considered when BRAMs embeds ECC engine, with single error correction and double error detection. The single error correction and double error detection events are notified by mean of two signals (*SBITERR* and *DBITERR* in Fig. 8).

Furthermore, in this special case, BRAMs can be equipped with single and double error injection signals (*INJECTSBITERR* and *INJECTDBITERR*), providing the chance of online testing ECC engine. Thus, the fault injector module can be further simplified, allowing the access from the Processing system, requiring 127 slices, only.

VI. Conclusions

This work presented a methodology for design time and online fast fault injection, suitable for safety critical SoPC. Two different case of study have been presented, highlighting pros and cons of their adoption, as well as some implementations details. Future work will investigate further methodology for fast online error injection on the real hardware to enhance the accessibility for testing in safety critical applications.

References

[1] Moore, G.E., "Cramming More Components Onto Integrated Circuits," Proceedings of the IEEE , vol.86, no.1, pp.82,85, January 1998

[2] Gadlage, M.J.; Eaton, P.H.; Benedetto, J.M.; Carts, M.; Zhu, V.; Turflinger, T.L., "Digital Device Error Rate Trends in Advanced CMOS Technologies," Nuclear Science, IEEE Transactions on , vol.53, no.6, pp.3466,3471, December 2006

[3] Gaspard, N.J.; Jagannathan, S.; Diggins, Z.J.; King, M.P.; Wen, S.-J.; Wong, R.; Loveless, T.D.; Lilja, K.; Bounasser, M.; Reece, T.; Witulski, A.F.; Holman, W.T.; Bhuva, B.L.; Massengill, L.W., "Technology Scaling Comparison of Flip-Flop Heavy-Ion Single-Event Upset Cross Sections," Nuclear Science, IEEE Transactions on , vol.60, no.6, pp.4368,4373, December 2013

[4] Oates, A.S.; Lin, M. H., "The scaling of electromigration lifetimes," Reliability Physics Symposium (IRPS), 2012 IEEE International , vol., no., pp.6B.2.1,6B.2.7, 15-19 April 2012

[5] International Electrotechnical Commission, "IEC 61508 Second Edition: Functional Safety of Electrical/Electronic/Programmable Electronic Systems", 2010.

[6] Mei-Chen Hsueh; Tsai, T.K.; Iyer, R.K., "Fault injection techniques and tools," Computer , vol.30, no.4, pp.75,82, Apr 1997

[7] Chakraborty, T.J.; Chen-Huan Chiang, "A novel fault injection method for system verification based on FPGA boundary scan architecture," Test Conference, 2002. Proceedings. International , vol., no., pp.923,929, 2002

[8] Portela-García, M.; López-Ongil, C.; Valderas, M.G.; Entrena, L., "Fault Injection in Modern Microprocessors Using On-Chip Debugging Infrastructures," Dependable and Secure Computing, IEEE Transactions on , vol.8, no.2, pp.308,314, March-April 2011

[9] Nazar, G.L.; Rech, P.; Frost, C.; Carro, L., "Radiation and Fault Injection Testing of a Fine-Grained Error Detection Technique for FPGAs," Nuclear Science, IEEE Transactions on , vol.60, no.4, pp.2742,2749, August 2013

[10] Tröger, P.; Salfner, Felix; Tschirpke, S., "Software-Implemented Fault Injection at Firmware Level," Dependability (DEPEND), 2010 Third International Conference on , vol., no., pp.13,16, 18-25 July 2010

[11] Benso, A.; Di Carlo, S.; Di Natale, G.; Prinetto, P.; Solcia, I.; Tagliaferri, L., "FAUST: fault-injection script-based tool," On-Line Testing Symposium, 2003. IOLTS 2003. 9th IEEE , vol., no., pp.160,, 7-9 July 2003

[12] D Gil, J Gracia, J.C Baraza, P.J Gil, Study, comparison and application of different VHDL-based fault injection techniques for the experimental validation of a fault-tolerant system, Microelectronics Journal, Volume 34, Issue 1, 1 January 2003

[13] Shirazi, M.S.; Morris, B.; Selvaraj, H., "Fast FPGA-based fault injection tool for embedded processors," Quality Electronic Design (ISQED), 2013 14th International Symposium on , vol., no., pp.476,480, 4-6 March 2013

[14] Grinschgl, Johannes; Krieg, Armin; Steger, C.; Weiss, R.; Bock, Holger; Haid, Josef, "Automatic saboteur placement for emulation-based multi-bit fault injection," Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2011 6th International Workshop on , vol., no., pp.1,8, 20-22 June 2011

[15] Azkarate-askasua, M.; Martinez, I.; Iturbe, X.; Obermaisser, R., "Dependability assessment of the time-triggered SoC prototype using FPGA fault injection," IECON 2011 - 37th Annual Conference on IEEE Industrial Electronics Society , vol., no., pp.2802,2807, 7-10 November 2011

[16] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J.-C. Fabre, J.-C. Laprie, E. Martins, D. Powell, "Fault Injection for Dependability Validation: A Methodology and Some Applications," IEEE Transactions on Software Engineering, vol. 16, no. 2, pp. 166-182, February, 1990

[17] Xilinx, "Zynq-7000 AP SoC Technical Reference Manual", UG585 (v1.7) February 11, 2014

[18] Ziade, Haissam, Rafic A. Ayoubi, and Raoul Velazco. "A survey on fault injection techniques." International Arab Journal Information Technology 1.2 (2004): 171-186.

[19] Benso, A. and Prinetto, P., "Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation" (1st ed.). Springer Publishing Company, Incorporated. 2010.

[20] Jun Xu; Ping Xu, "The Research of Memory Fault Simulation and Fault Injection Method for BIT Software Test," Instrumentation, Measurement, Computer, Communication and Control (IMCCC), 2012 Second International Conference on , vol., no., pp.718,722, 8-10 Dec. 2012

[21] Benso, A.; Prinetto, P.; Rebaudengo, M.; Reorda, M.S., "A fault injection environment for microprocessor-based boards," Test Conference, 1998. Proceedings., International , vol., no., pp.768,773, 18-23 Oct 1998

[22] Sonza Reorda, M.; Sterpone, L.; Violante, M.; Portela-Garcia, M.; Lopez-Ongil, C.; Entrena, L., "Fault Injection-based Reliability Evaluation of

SoPCs," Test Symposium, 2006. ETS '06. Eleventh IEEE European , vol., no., pp.75,82, 21-24 May 2006

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

[24] Ruano, O.; Maestro, J.A.; Reviriego, P., "Performance analysis and improvements for a simulation-based fault injection platform," Industrial Electronics, 2008. ISIE 2008. IEEE International Symposium on , vol., no., pp.2299,2304, June 30 2008-July 2 2008

[25] Pournaghdali, F.; Rajabzadeh, A.; Ahmadi, M., "VHDLSFI: A simulation-based multi-bit fault injection for dependability analysis," Computer and Knowledge Engineering (ICCKE), 2013 3th International eConference on , vol., no., pp.354,360, October 31 2013-November 1 2013

[26] Kwang-Ting Cheng; Shi-Yu Huang; Wei-Jin Dai, "Fault emulation: A new methodology for fault grading," Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on , vol.18, no.10, pp.1487,1495, October 1999

[27] P. Civera, L. Macchiarulo, M. Rebaudengo, M. Sonza Reorda, and M. Violante. 2002. An FPGA-Based Approach for Speeding-Up Fault Injection Campaigns on Safety-Critical Circuits. J. Electron. Test. 18, 3 (June 2002), 261-271

[28] Altera, "User-Customizable ARM-Based SoCs for Next-Generation Embedded Systems", WP-01167-1.1, June 2013

[29] Xilinx, "Zynq-7000 All Programmable SoC Overview", DS190 (v1.6), December 2013

[30] Microsemi Corp, "SmartFusion2 System-on-Chip FPGAs Product Brief", Rev.14, December 2013

[31] Vanhauwaert, P.; Leveugle, R.; Roche, P., "A Flexible SoPC-based Fault Injection Environment," Design and Diagnostics of Electronic Circuits and systems, 2006 IEEE , vol., no., pp.190,195, 18-21 April 2006

[32] Miklo, M.; Elks, C.R.; Williams, R.D., "Design of a high performance FPGA based fault injector for real-time safety-critical systems," Application-Specific Systems, Architectures and Processors (ASAP), 2011 IEEE International Conference on , vol., no., pp.243,246, 11-14 Sept. 2011

[33] Xilinx, "7 Series FPGAs Memory Resources", UG473 (v1.10) January 30, 2014

[34] Avnet, "ZedBoard (Zynq™ Evaluation and Development) Hardware User’s Guide", Version 2.2, January 2014

[35] Xilinx, "Vivado Design Suite User Guide", UG973 (v2013.4), January 2014

[36] Zedboard - SDK HelloWorld Example, avilable at http://zedboard.org/content/zedboard-sdk-helloworld-example