uc3m | Universidad Carlos III de Madrid

e-Archivo

This is a postprint version of the following published document:

Entrena, Luis; López-Ongil, Celia; García-Valderas, Mario; Portela-García, Marta; Nicolaidis, Michael. (2011). Hardware Fault Injection. In: Nicolaidis, M. (ed.) *Soft Errors in Modern Electronic Systems.* (Frontiers in Electronic Testing, 41). Springer. Pp. 141-166.

DOI: https://doi.org/10.1007/978-1-4419-6993-4_6

© Springer ScienceþBusiness Media, LLC 2011

Author's Proof

Chapter 6 1
Hardware Fault Injection 2

Luis Entrena, Celia López-Ongil, Mario García-Valderas, 3 AU1
Marta Portela-García, and Michael Nicolaidis 4
1 Electronic Technology Department, Carlos III University of Madrid, Spain. Email: { entrena, mgvalder, mportela, celia}@ing.uc3m.es
2 TIMA (CNRS, Grenoble INP, UJF), Grenoble-France,<michael.nicolaidis@imag.fr>

Uncorrected Proof

Hardware fault injection is the widely accepted approach to evaluate the behavior 5
of a circuit in the presence of faults. Thus, it plays a key role in the design of robust 6
circuits. This chapter presents a comprehensive review of hardware fault injection 7
techniques, including physical and logical approaches. The implementation of 8
effective fault injection systems is also analyzed. Particular emphasis is made 9
on the recently developed emulation-based techniques, which can provide large 10
flexibility along with unprecedented levels of performance. These capabilities 11
provide a way to tackle reliability evaluation of complex circuits. 12

6.1 Introduction 13

As technology progresses into nanometric scale, the concern for reliability is 14
growing. The introduction of new materials, processes, and novel devices along 15
with increasing complexity, power, performance, and die size affect reliability 16
negatively. On the contrary, the reduction in dimensions, capacitance, and voltage 17
results in less node critical charge, bringing up the soft-error threat. Actually, taking 18
into account all these trends, the soft-error rate (SER) per bit is expected to keep 19
stable, according to recent studies [1]. However, since the memory bit count and the 20
functionality integrated in logic components are increasing rapidly, the threat of 21
soft errors is becoming a reality for many applications where it was not a concern in 22
the past. The increasing use of electronic systems in safety critical applications, 23
where human life is at stake, forces to ensure dependability and makes it an important 24
challenge today. 25

C.L.-Ongil (✉)
e-mail: celia@ing.uc3m.es

M. Nicolaidis (ed.), *Soft Errors in Modern Electronic Systems*,
Frontiers in Electronic Testing 41, DOI 10.1007/978-1-4419-6993-4_6,
© Springer Science+Business Media, LLC 2011

Author's Proof

26 Providing quality of service in the presence of faults is the purpose of fault
27 tolerance. But before a fault-tolerant system is deployed, it must be tested and
28 validated. Thus, dependability evaluation plays an important role in the design of
29 fault-tolerant circuits. Fault injection, i.e., the deliberate injection of faults into a
30 circuit under test, is the widely accepted approach to evaluate fault tolerance. Fault
31 injection is intended to provide information about circuit reliability covering three
32 main goals: validate the circuit under test with respect to reliability requirements;
33 detect weak areas that require fault-tolerance enhancements; and forecast the
34 expected circuit behavior in the occurrence of faults.

35 From a general point of view, we can distinguish between hardware and
36 software fault injection, although the frontier between them is not well defined.
37 Software fault injection deals with software reliability and will not be treated
38 here. Hardware fault injection is related to hardware faults, which are generally
39 modeled at lower levels (e.g., logical or electrical) and are injected into a piece of
40 hardware.

41 In spite of the work made over many years, hardware fault injection is still a
42 challenging area. New types of faults and effects come to place or achieve
43 increasing relevance. In addition to permanent stuck-at faults or transient faults
44 affecting memory bits, such as single-event upsets (SEUs), today designers must
45 face the possibility of timing faults, single-event transients (SETs) affecting
46 combinational logic, and multiple bit upsets (MBUs) affecting memories. More
47 complex circuits need to be evaluated as a consequence of technology scaling and
48 increasing density. In particular, Systems on Chip (SoCs) include a variety of
49 components, such as microprocessors, memories, and peripherals, which pose
50 different fault injection requirements. The widespread use of field-programmable
51 technology confronts the need to evaluate the effect of errors on the configuration
52 bits. As complexity increases, the number of faults to be injected in order to
53 achieve statistical significance also increases. Thus, there is a need for new
54 approaches and solutions in order to accurately reproduce fault effects, increase
55 fault injection performance, and support the variety of existing technologies and
56 components.

57 This chapter summarizes the current state of the art in hardware fault injection
58 techniques and optimizations of the hardware fault injection process. It must be
59 noted that the fault injection process is not only concerned with the means to inject
60 faults. A complete environment is required for initialization of the circuit under test,
61 selection and application of appropriate workloads, collection of information about
62 faulty circuit behavior, comparison with the correct behavior, classification of fault
63 effects, and monitoring of the overall process. The importance of each of these tasks
64 must not be neglected, because all of them are relevant for a successful evaluation.

65 The remaining of the chapter is organized as follows. Section 6.2 reviews the
66 most relevant hardware fault injection techniques and the existing approaches to
67 inject faults. Section 6.3 describes the fault injection environments. Section 6.4
68 describes optimizations that contribute to increase fault injection performance.
69 Finally, Sect. 6.5 contains the conclusion of this chapter. AU2

Uncorrected Proof

Author's Proof

6.2 Hardware Fault Injection Techniques

Dependability evaluation of modern VLSI circuits entails the need of injecting realistic faults at internal locations, and observing in an efficient way the circuit behavior in the presence of these faults. Indeed, with the generalization of deep submicron technologies, there can be a much greater number of faults in digital systems, and a significant proportion of them occur internally in the chip [2]. There are different methods for injecting faults in integrated circuits mimicking hard or soft errors. Some of these methods have been used for several years, while others are being proposed in recent research works.

In every fault injection method, there are some common elements that may be defined at different abstractions levels. First of all, faults to be injected in the evaluation process should be selected. The possibility of provoking real faults within the device or just modeling the effect they cause (fault model) in the circuit elements must be considered. Second, the circuit to be checked, usually named device under test (DUT) or circuit under test (CUT), could be a commercial-off-the-shelf (COTS), a prototype, or a design model. The level of abstraction for the DUT is related to the type of fault to be injected. Third, a collection of workloads should be available in order to get a representative subset of the circuit functionality. Circuit robustness should be checked when working as closely as possible to normal operation in its final application. Finally, the most important element in every fault injection method is the expected result, which corresponds mainly to a measure of device robustness against faults.

Furthermore, any fault injection method requires a specific setup for the fault injection campaign. In particular, a dedicated PCB with the DUT must be developed. Also, a system for workload application and processing of the test results, including hardware, software, communication links, etc., should be implemented for results processing.

Faults are typically classified according to their duration into permanent, intermittent or transient faults. Permanent faults are related to manufacturing defects and circuit aging. Transient faults are mainly caused by the environment, such as cosmic radiation or electrical noise. They do not produce a permanent damage and their effects are known as soft errors.

Cosmic radiation is the main source of single-event effects (SEEs) in integrated circuits. SEEs are caused by single energetic particles and take many forms, with permanent or transient effects. Thus, in this case, injected faults range from real faults coming from natural cosmic radiation to fault models of SEEs; circuits to be tested range from COTS to design descriptions. The result of a fault injection campaign is the probability a device will fail when working. Typically, this measure is the failure in time (FIT), which stands for the number of circuit failures per 10^9 h, and it is referred to a given radiation environment.

The main hardware fault injection cases will be summarized in the following subsections. Some of them are also addressed in detail in other chapters.

Uncorrected Proof

Author's Proof

Uncorrected Proof

6.2.1 Physical Fault Injection

Physical fault injection methods use external perturbation sources, such as natural and accelerated particle radiation, laser beam, pin forcing, etc. The objective of this test is the analysis of circuit robustness in the presence of faults affecting a device. These methods are applied on COTS or prototypes for qualifying new technologies or existing chips for a new application environment.

These methods can cause a wide range of internal damages in the circuit under test: SEEs, displacement damage (DD) and total ionization dose (TID). Typically, studied SEEs are SEUs, SETs and single-event latchups (SELs).

6.2.1.1 Radiation Methods

The most traditional method for provoking internal soft errors in the circuit under test is the use of particle radiation. Cosmic radiation is the main source of SEEs in integrated circuits. Therefore, testing a device in its real environment (space, high altitude, etc.) is the most realistic way of evaluating its sensitivity with respect to SEEs. There are some practical disadvantages for this solution that are related to cost and time-to-market. Due to the low probability of error, weeks or months are generally required, as well as hundreds or thousands of samples, for obtaining valid measures. Another disadvantage is the unknown relationship between failures and the energy of particles striking the samples.

When shorter testing times and more controlled experiment setup are required, *accelerated radiation tests* are used for qualifying new technologies. The DUT receives a beam of particles, coming from a specific accelerator facility [3] or from a radioactive source [2]. In this case, few samples are needed (typically in the order of ten) as well as less time for testing (hours or days). Also, an energy or intensity sweep can be applied on the particle beam affecting the circuit under test. In this case, it is easy to know the SER with respect to the energy of particles.

In accelerated radiation tests, several types of particles are used for evaluating circuit robustness in harsh environments. Particles coming from cosmic radiation (primary or secondary radiation) are heavy ions, protons, and neutrons. Also, shells of devices emit alpha particles that provoke SEEs in the circuit.

The main origin of cosmic radiation is the sun [4]; it provokes ionizing particles (heavy ions and protons), known as primary radiation, in deep space and stratospheric orbits, and non-ionizing particles (neutrons), known as secondary radiation, in atmospheric applications. The IEEE standard for testing space-borne components [5] indicates the type of particles present in different environments. This information is used together with the final location of the circuit for stating the work environment and deciding the type of radiation test to be performed. When checking the dependability of circuits in aircrafts or in earth surface, neutron and alpha particles are used for accelerated radiation tests [6]. Alpha particles affect circuits, especially at earth surface. On the contrary, heavy

Author's Proof

ions are used for testing circuits working in nuclear or spatial applications [7]. 152
Finally, protons are employed for testing components of terrestrial satellites 153
[8, 9]. 154

Standards developed by space agencies [8, 9] or by JEDEC association [6, 7] 155
regulate radiation test procedures. In any case, simulation software tools are used 156
together with these tests in order to obtain a more accurate knowledge of damage 157
effect in devices. Material, type of particle, orbit, etc., are key elements in these 158
calculations. 159

In these experiments, the setup task is a heavy process. Flux, fluence, and energy 160
of particles must be set accurately (dosimetry is a key factor) for achieving a 161
significant number of events and avoiding TID damage. The final result obtained 162
is the cross-section, which is a function of particle energy or linear energy transfer 163
(LET) and gives the number of events detected with respect to the particle fluence 164
applied. 165

Static and dynamic tests should be performed on the DUT. While static tests 166
qualify the technology of a device, dynamic tests measure the robustness of a circuit 167
running in that device with a given workload. It is possible to disaggregate both 168
tests and avoid dynamic test under radiation. Velazco et al. [10] proved that 169
dynamic test results can be obtained by combining static test results (static cross- 170
section) and results obtained with another fault injection method. 171

The main disadvantages of fault injection based on accelerated radiation ground 172
testing are the high cost of the test campaigns and the relative small number of 173
events achieved per run, which may lead to results that are not statistically signifi- 174
cant. Also, controllability and observability are very limited. In any case, this type 175
of test is currently mandatory for qualifying a technology in aerospace applications. 176

6.2.1.2 Laser Methods 177

A relatively recent approach for injecting faults from an external source is the use of 178
laser beams. Laser incidence in the internal elements of the circuit causes effects 179
similar to the ones provoked by particles issued from cosmic radiation. Indeed, this 180
method is associated with bit-flip fault model for SEU effects. It is able to inject 181
faults in a very accurate way, with the help of a microscope and a laser beam spot 182
control. 183

There are research works [11–13] that prove the correlation between the results 184
obtained from accelerated radiation test and laser test. Although there is a 185
slight difference between the particle–material interaction and the photon–material 186
interaction, SERs obtained from laser beam exposure are commonly accepted 187
nowadays [13]. 188

Laser test provides a high level of accessibility to locate the circuit elements 189
where faults are injected. Also, this method implies less expensive equipment than 190
radiation ground test facilities and less complex experiment setup (e.g., it is not 191
necessary to separate the DUT in another PCB). 192

Uncorrected Proof

Author's Proof

6.2.1.3 Pin Forcing

Another solution for fault injection from external sources is pin forcing [14, 15]. It was proposed for testing relatively simple ICs. Some authors considered that forcing values at input/output pins of a device could provoke the same effect as SEEs in very simple circuits. There are several CAD tools developed for helping designers to execute fault injection campaigns, such as RIFLE [16], SCIFI [17], FIST [18], or Messaline[19]. Considering current complexities in ICs, this method is very limited. Currently, it is employed for testing other external aspects of reliability (vibrations, electrical noise, etc.), but it is not intended for SEU fault injection.

Although it is a really cheap solution, possible circuit damage due to values forced onto device pins, together with the poor controllability and observability provided in the increasingly complex ICs, makes this method unattractive for dependability analysis of current technologies.

Physical fault injection methods provide realistic measures of SERs, but they are very expensive. They are considered the best methods available for qualifying new technologies. Nevertheless, better solutions are required for testing circuits during the design process where re-design is possible and cheaper. Furthermore, very complex designs require testing large amounts of faults in order to obtain statistically valid results. When thinking of large fault sets for current designs, fault injection can be accomplished at higher abstraction levels.

6.2.2 Logical Fault Injection

Logical fault injection methods use logic resources of the circuit to access internal elements and insert the effect a fault provokes (fault model). These extra logic resources are originally intended for other purposes, such as the IEEE standard for Boundary Scan 1149.1 (JTAG) that provides an easy way for accessing internal scan path chains through a serial interface. Also, some commercial microprocessors include on-chip debugging (OCD) capabilities that enable access to internal memory elements (program counter, user registers, etc.). Finally, reconfiguration resources for programmable devices enable to control and observe internal configuration nodes and, therefore, injecting faults and observing their effects.

The undertaken fault models depend on the robustness analysis under execution. Therefore, bit-flip model is applied for SEUs and MBUs, stuck-at model is applied for permanent faults, voltage pulses are used for SETs, etc.

6.2.2.1 Software Implemented Fault Injection

SWIFI is intended for testing hardware by means of executing specific software that modifies internal memory elements (user accessible) according to a fault model.

Uncorrected Proof

Author's Proof

Fault injection can be performed during compilation time or during execution time [20]. In this last group, typical approaches use timers, such as FERRARI tool [21], or interruption routines, such as XCEPTION tool [22] or CEU tool [10]. More recently, new solutions have been presented [23] combining software-based techniques with previous approaches.

6.2.2.2 On-Chip Debugging for Microprocessors

Debugging resources provide direct access to internal registers, program counter, and other key elements in microprocessor architectures. This access can enable fault injection and fault effects observation in a rapid and effective way. Furthermore, the external accessibility of these capabilities makes the automation of fault injection campaigns easier. The use of OCD resources for testing purposes has been studied by some authors in recent years. FIMBUL tool uses JTAG interface for injecting bit-flip faults into memory elements of a microprocessor [17]. Rebaudengo et al. use the Motorola OCD, named background debugging mode (BDM), to execute fault injection through a serial port [24]. Also, NEXUS debugging standard is being used to enhance this fault injection method [25]. In [26] and [27], solutions are proposed by implementing specific hardware modules for interfacing between DUT (microprocessor) and host machine.

Recently, Portela et al. [28] have proposed another enhancement in the use of OCD capabilities, implementing in a hardware module the host in charge of injecting faults and analyzing obtained results. By reducing communication delays between hardware and software, fault injection process can be easily accelerated and automated.

6.2.2.3 Reconfiguration Resources

Reconfiguration resources in programmable devices make possible a direct fault injection within memory elements in prototyped designs. This method is widely used to evaluate the effect of faults in the configuration memory of FPGAs (field programmable gate arrays), which is a very important issue in these devices.

Partial reconfiguration reduces the time needed for performing fault injection in the configuration memory of FPGAs. Ref. [29] presents a solution based on reconfiguring by means of the Xilinx software JBits. In [30], a tool for injecting SEU faults in a Virtex® FPGA is proposed. This tool is able to inject faults in programmable interconnections, which are not accessible through commercial software tools (JBits). In recent contributions, Alderighi et al. proposed the FLIPPER fault injection platform which enables the fault-tolerance evaluation of hardened prototypes in FPGAs [31].

Uncorrected Proof

Author's Proof

265 ### 6.2.3 Logical Fault Injection by Circuit Emulation

266 As FPGA-based prototyping becomes popular for ASIC verification, it can also be
267 exploited for hardware fault injection. In this case, the circuit under test is proto-
268 typed in one or several FPGAs. This approach is generally known as emulation-
269 based fault injection. Contrary to the approaches mentioned in the previous section,
270 FPGAs are used here just as a means to support fault injection, and the final circuit
271 is to be implemented in some ASIC technology.

272 Fault injection in an FPGA-based prototype can take advantage from the flexi-
273 bility of field-programmable hardware. Fault injection requires high controllability
274 of each circuit node in order to modify its logic state. This can be obtained by using
275 the FPGA reconfiguration mechanisms to modify the circuit or the contents of
276 accessible memory elements. Another approach consists in inserting some addi-
277 tional hardware blocks in the prototype to support fault injection. These hardware
278 blocks are called *instruments*.

279 Emulation-based fault injection was originally developed for permanent faults.
280 In [32], a fault injection method is proposed for stuck-at faults. This method
281 consists in modifying the circuit by connecting a signal to a constant logic value.
282 Therefore, the FPGA must be resynthesized and reconfigured for each fault. Several
283 techniques to emulate faults in parallel are proposed to alleviate the resynthesis and
284 reconfiguration effort.

285 A fault injection technique for SEUs (bit-flips) based on run-time reconfigura-
286 tion of the FPGA is proposed in [33]. In this technique, flip-flop (FF) contents are
287 modified by controlling the asynchronous set/reset of each FF through the FPGA
288 configuration bitstream. Injection of a fault is performed with the following steps:
289 (a) at injection time, read the states of FFs; (b) reconfigure FPGA to set the
290 asynchronous set/reset switch of each FF as to keep the current state, except for
291 the faulty FF, that will be set in the opposite way; (c) pulse global set/reset line
292 (state of faulty FF is modified); and (d) reconfigure FPGA again to set the
293 asynchronous set/reset switches to the original value. The first step is performed
294 by readback of the configuration bitstream, which includes the states of FFs.
295 Readback is also used to check the results of each fault injection experiment and
296 classify fault effects. This idea is also followed up in the FT_UNSHADES
297 platform [34].

298 Note that this approach requires reconfiguring twice the FPGA for each fault.
299 Although partial reconfiguration can be used, the reconfiguration process is slow.
300 Fault injection rates range between 0.1 s and more than 1 s per fault, depending on
301 the length of the partial reconfiguration bitstream.

302 Circuit instrumentation is a means to overcome the limitations of FPGA recon-
303 figuration. It consists in inserting some pieces of hardware or instruments that can
304 provide external controllability and observability to inject a fault and observe its
305 effects. Then, the circuit is prototyped in an FPGA including the instruments. It is
306 important that circuit instrumentation can be automated in order to avoid handling
307 the circuit and to make the instrumentation process effective. On the contrary, the

Uncorrected Proof

Author's Proof

instruments should be small enough to obtain an acceptable overhead in the prototype.

In one of the earlier works [35], Hong et al. proposed a technique to avoid reconfiguring for each fault by adding some specialized hardware blocks. Each block is attached to a target node and contains a flip-flop that stores the injection signal value. These flip-flops are arranged in a chain, so that faults can be injected by shifting in the desired injection values in the chain.

A circuit instrumentation technique for the injection of non-permanent faults is proposed in [36]. This technique is intended to emulate SEUs by injecting faults in the circuit flip-flops. For this purpose, the circuit under test is modified by substituting each flip-flop by the instrument shown in Fig. 6.1. This instrument contains an additional flip-flop, called the mask flip-flop, and two gates that implement the mask logic. The mask flip-flop is used to select the fault injection target. At injection time, the inject signal is asserted to all instruments. Then, the input value at the nodes where the mask flip-flop is set will flip and the fault is injected. More elaborated instruments have been proposed by Lopez et al. [37] that will be described in Sect. 6.4.

The mask flip-flops are arranged in a scan chain that can be loaded serially. Once the mask scan chain is loaded, the inject signal is asserted at the required time to inject a fault. Faults can be injected at different nodes by shifting in the mask scan chain. MBUs are supported by setting more than 1 bit in the mask. Eventually, the contents of the functional flip-flops can be loaded into the mask flip-flops. This operation captures the internal state of the circuit, which can be observed by shifting out the mask chain.

The circuit instrumentation technique is very efficient, as it does not require reconfiguring the circuit for every fault. Also, setting the fault injection mask is much faster than FPGA reconfiguration. In addition, latent faults can be detected by checking the circuit state as obtained through the scan chain after the fault injection process. Experimental results [36] show a fault injection rate of 10,000 faults/s by

Uncorrected Proof

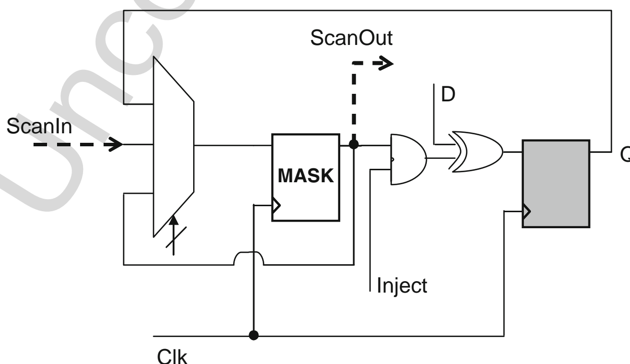


Fig. 6.1 Fault injection instrument in [36]

Author's Proof

337 using a 20-MHz clock and a short workload (100 test vectors). For large workloads,
338 the fault injection rate is inversely proportional to the workload length.
339 Injecting and propagating SETs is much more difficult, since it requires proto-
340 typing the logic delays of the circuit under test. Synthesizing the circuit under test
341 for an FPGA would produce an equivalent functional circuit model, but with
342 different gate delays. Existing approaches for SET emulation are based on embed-
343 ding timing information in some way, such as the topology of the circuit [38].
344 Recently, an efficient approach has been proposed using a quantized delay model
345 [39]. In this model, gate delays are rounded to a multiple of a small amount of
346 time, addressed as time quantum. Quantized delays can be implemented in an
347 FPGA using shift registers where the time quantum corresponds to a clock cycle.
348 Experimental results show a fault injection rate in excess of one million faults/s,
349 representing an improvement of three orders of magnitude with respect to a
350 simulation-based approach.
351 The flexibility provided by FPGAs can be used to support some fault injection
352 functions. For example, the FPGA prototype can include two instances of the circuit
353 under test that are dedicated, respectively, to prototype the golden (fault-free) and
354 the faulty circuit. Both instances run in parallel and the outputs can be compared
355 inside the FPGA at the end of the execution to detect failures [34, 40]. A refined
356 solution consists in duplicating just the sequential elements, sharing the combina-
357 tional logic [37]. In this case, the golden and faulty instances run in alternate clock
358 cycles.
359 Most of the work on emulation-based hardware fault injection focuses on
360 general-purpose logic, but circuits may also include embedded memories. As the
361 number of storage elements in memories is generally very large, memories are in
362 fact very relevant. However, the controllability and observability are limited to a
363 memory access per memory port and clock cycle.
364 Memories can be emulated by forcing the synthesizer to treat them as flip-flops. AU4
365 This approach would usually produce emulation circuits much larger than commer-
366 cially available FPGAs. Therefore, embedded memories must be instrumented in a
367 particular way to support fault injection.
368 A memory can be implemented in an FPGA using FPGA memory blocks. In this
369 case, fault injection is performed by instrumenting the memory buses [28]. In [40],
370 memories are implemented using dual-port FPGA memory blocks, where one of the
371 ports is specifically devoted to fault injection. At the fault injection time, the target
372 memory position is read, XORed with the fault mask, and then written back. A
373 monitor circuit is included to clear all the internal memory after an experiment, in
374 order to avoid accumulation of errors, and to read serially all the internal memory
375 in order to compare it and detect faults. These two operations take a lot of time, just
376 in proportion to memory size, and must be performed for every fault injection
377 experiment. Thus, injecting a fault in a memory position can be achieved by
378 instrumenting the memory buses, but the initialization of the memory and the
379 extraction and comparison of the fault injection results for analysis are the major
380 problems of memory fault injection.

Uncorrected Proof

Author's Proof

6.3 Fault Injection System

A hardware fault injection system is able to execute a circuit with a workload in the presence of faults, and compare the faulty behavior with the fault-free behavior. A fault injection system is typically composed of the following elements:

- The CUT
- A fault injection mechanism, which can be physical or logical
- A test environment, in charge of the following tasks:
  - Supply the vectors required for the workload
  - Check the effect of faults in the CUT
  - Collect results
  - Classify faults
  - Control the whole process

Fault injection systems are used to perform fault injection campaigns, which are experiments intended for obtaining a measurement of the circuit reliability. Some examples of this measurement are the fault dictionary, the circuit cross-section, the SER, the mean time between failures, etc.

Fault injection systems using physical fault injection require a prototype of the DUT, which is exposed to the fault provoking element (radiation, laser). The system has to be built in such a way that the DUT is correctly exposed, but the rest of the system is not affected by the fault source. In accelerated radiation experiments, the CUT must be separated from the rest of the system, in order to receive the beam without affecting the test environment. For example, the THESIC system [41] consists of a motherboard as test environment, and a mezzanine board for the CUT. For laser campaign, the circuit package must be removed for the laser beam to be effective, and the CUT must also be visible through a microscope to locate the laser incidence point [42].

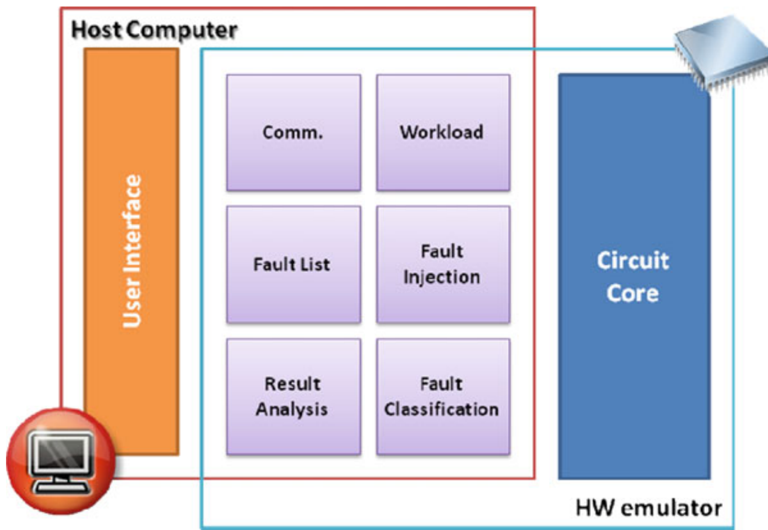
Systems using logical fault injection have a similar structure but use a different fault injection mechanism, so there are no restrictions related to physical exposure aspects. The only additional part to add is a fault injection method, for example, a host-controlled JTAG interface connected to the OCD of the CUT.

The rest of this section will cover the implementation of systems using logical fault injection by circuit emulation. This kind of system uses FPGA-based prototyping to implement the CUT. As this is the most general and flexible scenario, there is a large variety of solutions. Building these systems represent a very challenging task as far as the process performance is concerned.

There is a wide range of possibilities to build emulation-based fault injection systems, as there are many tasks that can be executed in a host computer or in the hardware. Fig. 6.2 shows the main required tasks. The user interaction takes place at the host computer and the circuit emulation is performed in the hardware (circuit core). The rest of the tasks may be executed either by the host computer or by the hardware.

Uncorrected Proof

Author's Proof



This figure will be printed in b/w

Fig. 6.2 Emulation-based fault injection system components

422 There are a lot of intermediate possibilities, depending on the tasks assigned to
423 the host computer and to the hardware. The speed of the communication channel is
424 critical when considering the amount of information to transfer.
425 In general, the tasks required to perform a fault injection campaign comprise
426 fault list management, workload application, fault injection, fault classification and
427 result analysis. These tasks are analyzed in the following paragraphs.

428 *6.3.1 Workload*

429 A workload must be provided to the circuit under evaluation for execution. The
430 implementation of this task must consider a trade-off between flexibility, perfor-
431 mance, and resource usage. Several approaches can be considered. Test vectors can
432 be generated at the host computer and sent to the hardware when they are going to
433 be applied. This method is very flexible, as the workload can be changed very
434 easily, but implies a continuous host–hardware communication which slows down
435 the execution.
436 On the contrary, a stimulus generation block can be implemented in the hard-
437 ware, next to the circuit core, so it can supply vectors at the speed they are required.
438 A simple approach is to use some BIST-like vector generation circuit, like an
439 LFSR. This kind of implementation is very fast and uses very little resources, but
440 the workload obtained may not be very representative.
441 An intermediate solution is to store test vectors in the FPGA memory. This
442 solution is flexible, as the workload can be easily changed by downloading a new

Uncorrected Proof

Author's Proof

one or by reconfiguring the FPGA, and it is also fast, because test vectors are fed to the circuit core by hardware. However, FPGA devices usually have a limited amount of internal memory, representing a drawback for this method. In order to improve resource usage, test vectors may be compressed, or external memory may be used if it is available on the FPGA board.

6.3.2 Fault List

The fault list management task has some important aspects, both in design and implementation. The ideal case is to generate a fault list including faults at every location and every time instant, for a given workload. Considering bit-flip fault model for SEUs, the complete single fault list would include a fault in every memory element and every clock cycle of the workload. This approach is practical only if the circuit is either very small or the fault injection system is very efficient, like Autonomous Emulation [37].

Usually, the system is not so efficient to perform a fault injection campaign with all possible faults in a reasonable time. In these cases, the fault space must be sampled to obtain a statistically representative subset. There are several approaches to create the fault list. The simplest one is to use random fault list generation, both in fault localization and time instant, although it must be taken into account that a computer or a hardware generated list is not really random, but pseudo-random. Other proposals include the use of Poisson distribution for the generation of the time instants, in order to reproduce the results of a radiation experiment. A deeper discussion on these aspects can be found in [43].

Regarding implementation, the fault list can be generated at the host computer or in the hardware. If it is generated at the host computer, it must then be transferred to the hardware. It is advisable to implement an intermediate storage mechanism, so that the emulator does not need to wait for the new fault when it has already finished processing the previous one. This mechanism can make use of internal FPGA memory or on-board memory.

In this case, the impact in performance is not as high as for the workload case. For the workload, a test vector is required every clock cycle, but a new fault is required only when the previous one has already been processed.

6.3.3 Fault Classification

The classification of a fault can be made out of the comparison of the faulty and the golden executions.

If a fault is injected and after some time it produces a result different than expected, it is called a *failure*. If the fault effect completely disappears from the circuit after some execution time, the fault is called *silent*. If the faulty circuit shows

Uncorrected Proof

Author's Proof

differences with the golden one after the execution of the workload, but it did not produce any error in the results, the fault is called *latent*. If the circuit has some built-in fault detection mechanism, the faults can also be classified as *detected* or *not detected*. In microprocessor-based circuits, a fault could also be classified as *lost of sequence*, when the effect of the fault is modifying the normal instruction sequence, preventing the circuit to reach the end of the workload.

Concerning failures, the condition of producing an erroneous result is different for every case. For a control circuit, an error can be a value in the outputs that is different than expected. For an algorithm processor, the calculation results must be checked and they can be written at the circuit outputs or stored in memory. Failures could also be sub-classified by a criticality-based criteria. For example, some errors could produce physical damage in the system (e.g., an electronically controlled mechanical engine) or produce a dangerous situation (a brake system), while others may be irrelevant (a wrong pixel in an image).

Latent faults represent an additional problem. The system must have a mechanism to compare the golden and the faulty circuit states to decide if the fault is still present. For example, two instances of the circuit can be implemented, and a mechanism to compare them must be included. If partial reconfiguration is used, readback of the circuit flip-flops can be used for this purpose with a high penalty on performance [34]. With instrumented circuit technique, flip-flops are duplicated and compared [37], so that latent faults can be detected online.

Performance will profit from an early fault classification mechanism. Being able to stop the execution immediately after the fault is classified will allow saving some time. If the classification is made in hardware, it is easier to implement a fast classification mechanism. This aspect will be explained in more detail in the next section.

6.3.4 Result Analysis

The results of a fault injection campaign can be expressed in several ways. In terms of circuit qualification, it is usually required to obtain a single figure for the circuit reliability, like the SER, expressed by either number of FIT, or Mean Time Between Failures (MTBF). These figures are calculated using information from fault injection campaigns and taking into account the environment where the circuit will operate. Emulation-based fault injection campaigns are useful to obtain information about the consequences of faults. Information about fault occurrence probabilities and other aspects must be obtained using other methods.

The most complete result information that can be obtained from a fault injection campaign is the fault dictionary, which is the list that holds the classification of every injected fault. The complete fault dictionary is very useful to locate weak areas in the circuit or critical tasks in the workload.

The implementation problems for this task are quite similar to those of the fault list. A new result is generated for every processed fault. Results can be transferred

Uncorrected Proof

Author's Proof

immediately after processing to the host, or they can be temporally stored in the hardware to improve performance.

In case result generation is very fast or there is no temporal storage available, statistical measures may be collected in the hardware. For example, results can be classified per location, or per injection instant, or just percentages can be calculated.

6.3.5 Communication

Emulator–host communication has a great impact on the system performance. In order to improve the performance, we can either increase the speed of the communication channel or decrease the amount or the frequency of the transmitted data.

Commonly used communication mechanisms are serial ports, USB, Ethernet, or PCI.

Obviously, an increment in the channel speed will result in an overall speed improvement, but several aspects must be taken into account. For example, communication channels like USB can be very fast, but only in burst mode, transmitting big amounts of data in a single pack.

In the case of a fault injection system, the information to transmit (test vectors, fault list, fault dictionary, and control commands) can be sparse in time. This approach is not very efficient for high speed communication channels, so it is advisable to design the emulator, including data compaction or communication buffers. In order to obtain the maximum performance, the objective is to maintain the core emulation circuit running as much time as possible, and avoid the time gaps due to communication.

6.4 Fault Injection Optimizations

Emulation-based fault injection techniques have the capability of notably speeding up the fault-tolerance evaluation process regarding other methods. Injecting millions of faults in a few hours is possible using these techniques. However, reducing this time is a very interesting goal, since fault-tolerance evaluation is a task performed many times during the circuit development. Furthermore, current circuits have large complexity, including an increasing count of sensitive areas that can be affected by faults. Therefore, the higher the number of possible faults, the higher the number of faults that must be injected to obtain a significant measurement of the circuit robustness. Definitely, speeding up the fault-tolerance evaluation process is required.

Several approaches exist to speed up the fault injection process using emulation-based techniques. In the following sections, we will describe the main sources of fault injection inefficiency and solutions to overcome them.

Uncorrected Proof

Author's Proof

557 ### 6.4.1 Autonomous Emulation

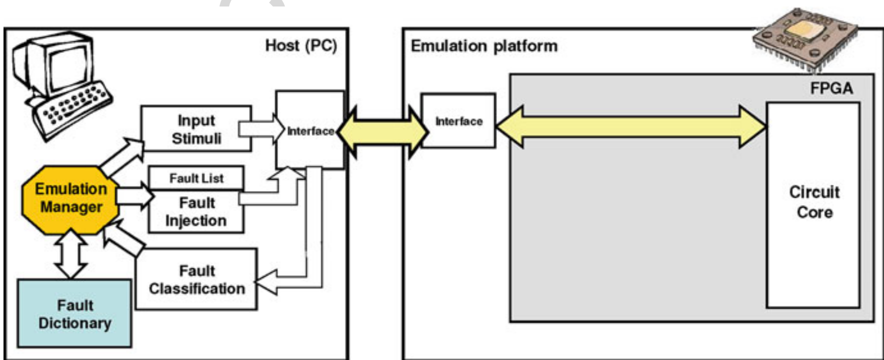
558 Emulation-based techniques profit from the capability of an FPGA to emulate
559 circuit behavior at hardware speed. Using typical emulation-based fault injection
560 solutions, the emulation process is interrupted every time the emulator needs to wait
561 for the host to apply the stimuli, to inject a fault, or to check the output values. Then,
562 a very intensive interaction is required between the emulator and the host computer.
563 The host controls the injection and evaluation of every fault. This introduces a
564 performance bottleneck due to the communication between the emulator and the
565 host computer, which prevents taking full advantage of the FPGA capabilities for
566 fast hardware emulation.

567 Autonomous emulation [37] is a fault injection solution aimed at avoiding the
568 intensive communication between host and emulation platform. It consists in
569 implementing the whole injection system in the FPGA by making use of the
570 instrumentation mechanism to insert faults in the circuit under test (Fig. 6.4). The
571 FPGA is in charge of performing the following tasks:

572 1. Managing the whole fault injection process.
573 2. Applying the input stimuli to the circuit under test.
574 3. Activating the fault injection.
575 4. Watching the circuit behavior under faults and classifying the injected fault
576 depending on its effect on the circuit functionality.

577 Using autonomous emulation, access to any circuit-sensitive element is simple
578 and straightforward, and the required time necessary to perform the different
579 injection tasks can be significantly reduced. Figure 6.3 shows the typical emula-
580 tion-based solutions scheme and Fig. 6.4 the autonomous emulation scheme with
581 the purpose of illustrating the differences between both systems. The autonomous
582 emulation system benefits from the available resources in current FPGA platforms,
583 like memory blocks, in order to implement more tasks close to the circuit under test,
584 which minimizes the required interaction with the host computer. The enhance-
585 ments provided by the autonomous emulation solution are as follows:

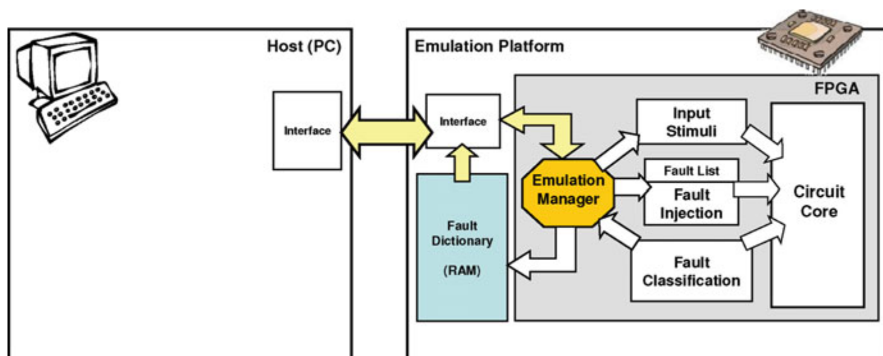
Uncorrected Proof



This figure will be printed in b/w

Fig. 6.3 Typical scheme for an emulation-based fault injection system

Author's Proof



This figure will be printed in b/w

Fig. 6.4 Autonomous Emulation scheme

- The required communication between host computer and the emulation platform is minimized, being established only twice, at the beginning of the evaluation process to configure the FPGA from the PC, and at the end of the fault injection campaign to collect the obtained results, that is, to download the fault dictionary.
- Observability and controllability are significantly enhanced, since access to the memory elements does not require a particular communication channel and then it is straightforward and easier. In general, the typical emulation-based techniques set a trade-off between the process speed and the observability of the internal circuit resources, because higher observability requires more information exchange and, therefore, more time to spend in the evaluation process. The autonomous emulation system provides a high observability without penalty in the injection process speed, since the injection system and the circuit under test are implemented in the same device.
- Hardware implementation makes the parallel execution of different injection tasks and it speeds up the whole process with respect to a software implementation.

Once the PC-FPGA communication, that is, the main limitation in fault emulation techniques, has been minimized, the fault injection process is much more efficient. Moreover, the access to the circuit internal resources does not require exchange of information between the host computer and the FPGA. This feature allows the application of new optimizations to reduce the time spent per fault.

Uncorrected Proof

6.4.2 Fault Evaluation Process

The fault injection process can be optimized by applying techniques to reduce the time spent in the different steps needed to evaluate the consequences of a fault. For a given workload these steps are the following (Fig. 6.5):

Author's Proof

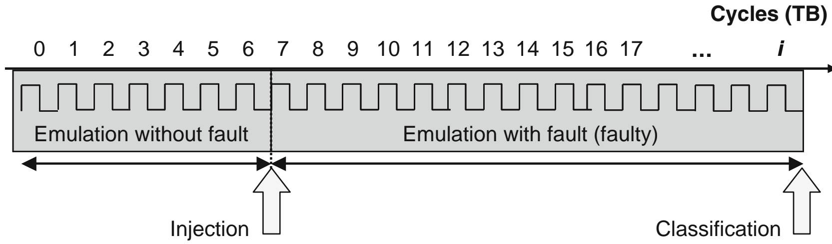


Fig. 6.5 Time spent to emulate each fault

611 1. Reach the circuit state corresponding to the injection instant. The most common
612 way to do it consists of running the workload from the beginning until the
613 injection instant.
614 2. Inject the fault.
615 3. Classify the fault according to its effect on the circuit behavior. For this purpose,
616 the circuit under test resumes the workload execution since the injection instant
617 until the fault is classified or the workload finishes.

618 In the worst case, emulating the complete workload for each fault can be
619 required. A fault injection campaign with a large number of injected faults and
620 long workloads may involve excessive time to complete the evaluation process.
621 With the instrumentation-based mechanism, the fault injection task takes just one
622 clock cycle and so possible time optimization should be applied to the other steps
623 (1 and 3). The time required for reaching the circuit state at the injection instant can
624 be optimized by applying techniques to save fault-free emulation time. A solution
625 consists in doing a previous storage of the circuit state and a posterior reloading of
626 this state in the next fault injection (state restoration). Regarding fault classification,
627 techniques can be applied to speed up fault emulation by aborting execution as soon
628 as the fault can be classified, profiting from the higher observability available in an
629 Autonomous Emulation system. In the following sections, these optimizations are
630 detailed for SEU faults.

631 *6.4.3 State Restoration*

632 The circuit under test can get to the state corresponding to the injection instant in
633 two different ways:

634 • Emulating the workload until the injection instant is reached.
635 • Storing the required state in memory elements of the circuit and restoring it just
636 before the fault injection instant (state restoration). In this case, additional
637 hardware to store the corresponding state is necessary.

638 State restoration avoids the fault-free circuit emulation for every injected fault.
639 Required states are easily obtained from the golden execution, run just once.

Uncorrected Proof

Author's Proof

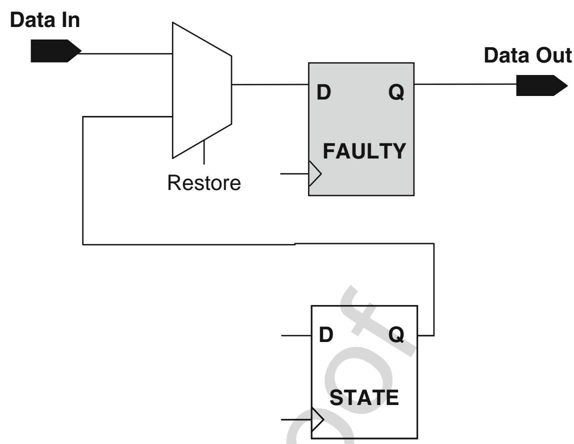


Fig. 6.6 Possible instrument to support injection state restoration in one clock cycle

Uncorrected Proof

The obtained benefit will depend on the time required to perform the restoration and, therefore, on the technique used to implement this optimization. Figure 6.6 presents a possible scheme to replace every original flip-flop in order to support the state restoration in just one clock cycle. It includes an additional flip-flop that contains the injection state to be restored in the circuit when the fault is going to be emulated.

Let us suppose that the state restoration requires only one clock cycle, the workload consists of C clock cycles, and the circuit under test contains F sensitive memory elements. Considering that all the memory elements have the same probability to be affected by a fault in any workload cycle, the number of possible single faults is $F \cdot C$. In the worst case, with no optimizations, C clock cycles are necessary to emulate each fault. Taking into account all the possible single faults, the total time spent during the fault injection campaign in emulating the circuit without faults is $(1 + 2 + 3 + \cdots + C - 1)$ clock cycles. When the state restoration is performed in just one cycle, this optimization avoids the emulation of C_S clock cycles, where

$$C_S = F\frac{C(C-1)}{2}$$

For example, for a circuit with $F = 10^3$, a workload with $C = 10^5$ clock cycles, and $C_S \sim 0.5 \times 10^{13}$ clock cycles, the total saved time at 100 MHz would be 14 h.

6.4.4 *Early Fault Classification*

Emulation of a fault finishes when the fault is classified or when the end of the workload is reached. A typical fault classification consists in considering three

Author's Proof

661 categories: failure, latent, and silent faults. In order to detect a failure, the faulty
662 circuit outputs must be compared with the golden behavior. To distinguish between
663 silent and latent faults, internal resources must be observed. Reducing the time to
664 classify faults introduces an important optimization in the evaluation process speed.
665 In general, fault injection techniques stop the fault evaluation as soon as a failure is
666 detected, since outputs observation and comparison with expected values are
667 usually straightforward. Due to the limited observability of the internal resources,
668 in most of the hardware fault injection techniques, classifying a fault as either silent
669 or latent is not feasible or, otherwise the classification is performed at the end of the
670 workload execution, which is very time-consuming. However, silent faults can be
671 detected as soon as the fault effect disappears if the internal elements are observed
672 continuously.

AU6

673 Speeding up silent fault classification requires access to every memory element
674 within the circuit under test in a fast and continuous way, comparing their content
675 with the golden circuit state. Additional hardware is used to store the golden state
676 and to perform the comparison. This extra hardware is shown in Fig. 6.7 and
677 consists of two flip-flops to run the golden and the faulty execution at every
678 workload instant. This optimization is possible in an Autonomous Emulation
679 system with a low cost, since the complete system is implemented in the same
680 hardware device.

681 Early silent fault classification enhances the fault injection process speed,
682 especially in circuits with fault-tolerant structures that correct or mask faults,
683 where the percentage of silent faults is high. Therefore, applying both optimiza-
684 tions, state restoration (described in the previous section) and early silent fault
685 classification, the time spent in emulating one fault can be drastically reduced
686 (Fig. 6.8).

687 Putting it all together, [37] describes an instrument to replace every original flip-
688 flop that supports Autonomous Emulation, state restoration, and early silent fault
689 classification. Such instrument is shown in Fig. 6.9. In this case, combinational
690 logic is shared, avoiding the duplication of the complete circuit. Then, the faulty
691 and golden emulation are executed alternately. This implementation for an Auton-
692 omous Emulation system is named *Time-Multiplexed* technique.

Uncorrected Proof

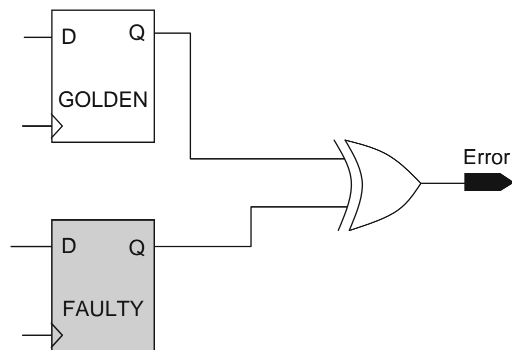


Fig. 6.7 Additional hardware required to implement early fault classification

Author's Proof

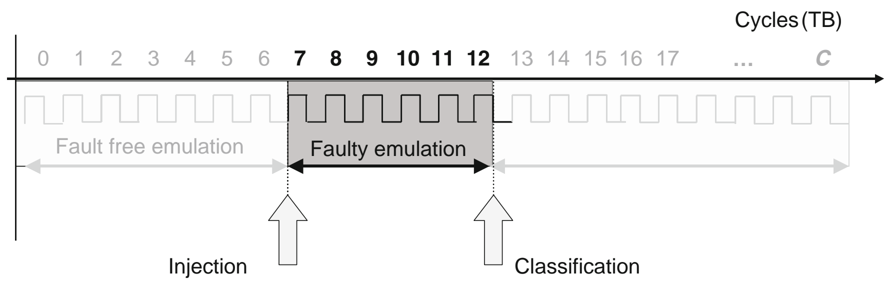


Fig. 6.8 Optimized fault emulation

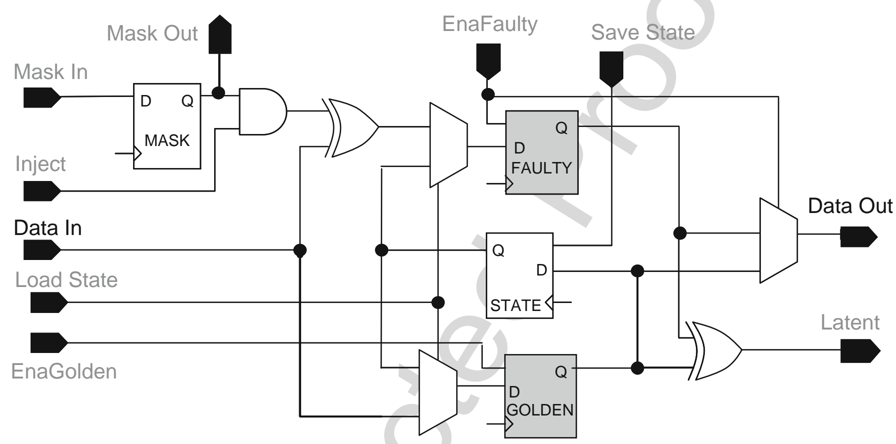


Fig. 6.9 Hardware logic to support all optimizations presented in [37]

Uncorrected Proof

For failure and silent faults, the time elapsed between fault injection and fault classification is usually a few clock cycles in circuits with fault-tolerant mechanisms. Only faults with long latencies require the execution of most of the workload, in case of latent faults until the end of the workload. Therefore, if fault latencies and the number of latent faults are small, which is the usual case in a well-defined experiment, the reduction in execution time is proportional to the workload length. 693 694 695 696 697 698

Experimental results reported in [37] show that using the described optimizations, the fault-tolerance evaluation process can achieve fault injection rates in the order of one million faults per second. 699 700 701

6.4.5 Embedded Memories 702

Embedded memories are common components in modern digital circuits. SRAM memories are sensitive to SEU in the same manner as flip-flops, and then, fault 703 704

Author's Proof

705 injection must be performed not only to evaluate flip-flops but also to evaluate
706 effects in the circuit behavior when an SEU affects a memory cell.
707 Fault injection in embedded memories using emulation-based techniques is a
708 complex task due to the limited observability of the memory cells, since only a
709 memory word can be accessed in a clock cycle. For example, in order to know if
710 there is an error in a memory, the complete memory content should be read, which
711 is very time-consuming.
712 Few works have been published about emulation-based fault injection in circuits
713 with embedded memories [28, 40, 44, 45]. Civera et al. [44] proposed the fault
714 injection in memory controlling the control and data memory buses to insert a fault
715 in a given memory bit, but it does not propose a solution to analyze faults inside the
716 memory. Lima et al. [40] presents a memory model that consists of a dual-port
717 memory; one port is used to perform the golden emulation, while the other port is
718 used to inject faults. However, the result analysis is very time-consuming since it
719 consists in reading every memory word, comparing obtained data to expected ones.
720 Portela-García et al. [28] describes an Autonomous Emulation system with
721 optimizations in a circuit with embedded memories. Autonomous Emulation speeds
722 up the injection process by minimizing PC-FPGA communication requirements and
723 including optimizations previously described (state restoration and early silent fault
724 classification). In order to apply the Autonomous Emulation concept in complex
725 circuits, solving the fault injection in embedded memories in a fast and cost-
726 effective way is mandatory.
727 Autonomous Emulation solution is based in an instrumentation mechanism, so a
728 memory instrument is necessary. It is assumed that embedded memories are
729 synchronous, i.e., they can be prototyped in current FPGAs using the available
730 block RAMs components, and they do not contain useful information before
731 starting the execution. The objective is to achieve a memory model that supports
732 state restoration and early silent fault classification [45].
733 The proposed solution is based on controlling and observing memory access
734 buses (address, data, and control signals). On the one hand, fault classification
735 requires distinguishing between silent and latent faults. For this purpose, input data
736 bus and control access signals (like write enable, output enable, etc.) in golden and
737 faulty execution are compared (Fig. 6.10). The emulation controller detects the
738 insertion of faulty data in the memory and checks if fault effect is cancelled by
739 writing the correct data during workload execution. As soon as the fault disappears,
740 it is classified as silent.
741 On the other hand, fault injection to emulate an SEU in a memory cell is
742 performed only in read data, since faults in other memory words do not affect the
743 circuit behavior and they would be classified as latent faults. Therefore, the number
744 of faults to evaluate is reduced significantly.
745 A possible implementation of the faulty memory in this model consists in storing
746 just the faulty memory words. In order to access Faulty Memory in a fast way,
747 spending just one clock cycle, it is implemented using a Content Addressable
748 Memory (CAM) [45]. Therefore, the faulty memory contains the addresses that
749 store a fault and the faulty data itself. If a given address is stored in the faulty

Uncorrected Proof

Author's Proof

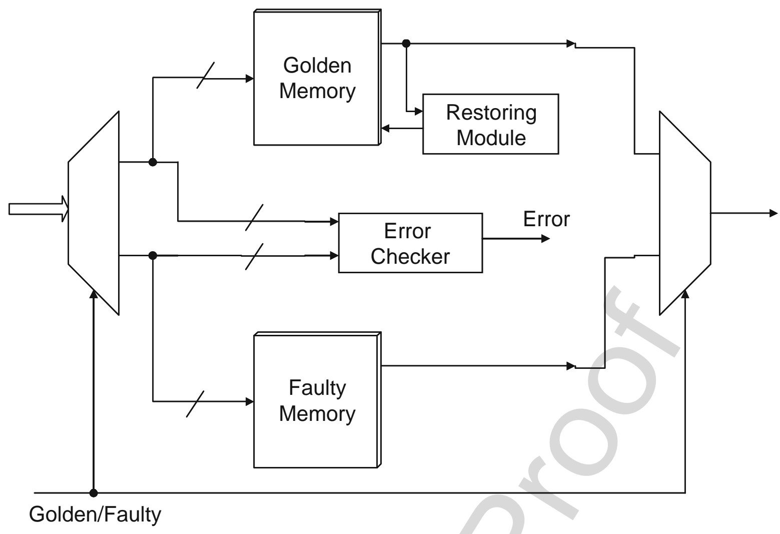


Fig. 6.10 Memory instrument to support Autonomous Emulation in complex circuits with embedded memories

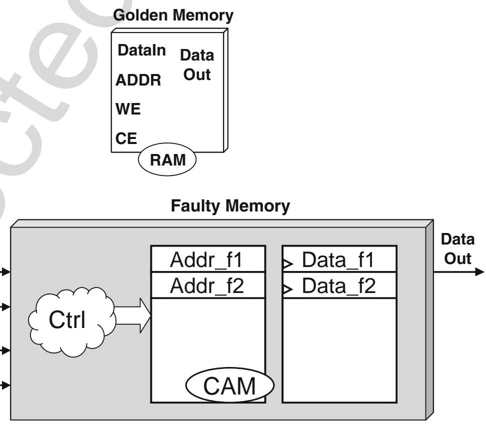


Fig. 6.11 ECAM model is a possible implementation for the memory instrument compliant to Autonomous Emulation

Uncorrected Proof

memory, the corresponding memory word contains a fault. Otherwise, the data are stored only in golden memory. This implementation is named Error Content Addressable Memory (ECAM), see Fig. 6.11. ECAM implementation is very suitable to perform the required Autonomous Emulation tasks, such as state restoration or silent fault classification (if faulty memory is empty).

The size of an ECAM implementation fixes the maximum number of errors that can be considered. In practice, the probability that N faults in memory are cancelled (writing a correct value) is very low for just a small value of N. Therefore,

Author's Proof

758 this solution implies less area overhead than other possible implementations
759 (like memory duplication).
760 Portela-García et al. [28] and Valderas et al. [46] present experimental results on
761 a LEON2 microprocessor. The experiments consist in injecting millions of faults in
762 flip-flops and memories by means of an Autonomous Emulation system.

763 6.5 Conclusions

764 Hardware fault injection plays a key role in the design and validation of robust
765 circuits. As hardware reliability is becoming an increasing concern in many appli-
766 cation areas, there is a need for new approaches and solutions that can deal with
767 more complex circuits and reproduce fault effects accurately and efficiently.
768 Hardware fault injection methods have significantly evolved in the last years.
769 Among the physical fault injection methods, accelerated radiation tests are the most
770 used, but laser fault injection has gone through substantial developments. On the
771 contrary, FPGAs can support very efficient logical fault injection methods, such as
772 Autonomous Emulation. These methods can provide unprecedented levels of per-
773 formance and fault injection capabilities, and represent suitable fault injection
774 mechanisms to complement physical methods.

Uncorrected Proof

775 References

776 1. R. C. Baumann, "Radiation-Induced Soft Errors in Advanced Semiconductor Technologies",
777 IEEE Transactions on Device and Materials Reliability, Vol. 5, No. 3, pp. 305–316, September
778 2005.
779 2. J. Karlsson, P. Liden, P. Dalgren, R. Johansson, U. Gunnelfo, "Using Heavy-Ion Radiation to
780 Validate Fault Handling Mechanisms", IEEE Micro, pp. 8–23, February 1994.
781 3. S. Duzellier, G. Berger, "Test Facilities for SEE and Dose Testing", Radiation Effects on
782 Embedded Systems. Springer 2007. The Netherlands. pp. 201–232.
783 4. R. Ecoffet, "In-Flight Anomalies on Electronic Devices", Radiation Effects on Embedded
784 Systems. Springer 2007. The Netherlands. pp. 31–68.
785 5. IEEE Standard for Environmental Specifications for Spaceborne Computer Modules,
786 March 1997.
787 6. JEDEC Standard JESD89A, "Measurement and Reporting of Alpha Particle and Terrestrial
788 Cosmic Ray-Induced Soft Errors in Semiconductor Devices", October 2006.
789 7. JEDEC Standard JESD57, "Test Procedures for the Measurement of Single-Event Effects
790 in Semiconductor Devices from Heavy Ion Irradiation", December 1996.
791 8. S. Buchner, P. Marshall, S. Kniffin, K. LaBel, "Proton Test Guideline Development – Lessons
792 Learned", NASA/Goddard Space Flight Center, NEPP, August 2002.
793 9. European Space Agency, "Single Event Effects Test Method and Guidelines", October 1995.
794 10. R. Velazco, S. Rezgui, R. Ecoffet, "Predicting Error Rate for Microprocessor-Based Digital
795 Architectures Through C.E.U. (Code Emulating Upsets) Injection", IEEE Transactions on
796 Nuclear Science, Vol. 47, No. 6, pp. 2405–2411, December 2000.
797 11. R. Velazco, B. Martinet, G. Auvert, "Laser Injection of Spot Effects on Integrated Circuits",
798 1st Asian Test Symposium, pp. 158–163, November 1992.

Author's Proof

12. P. Fouillat, V. Pouget, D. Lewis, S. Buchner, D. McMorrow, “Investigation of Single-Event Transients in Fast Integrated Circuits with a Pulsed Laser”, International Journal of High Speed Electronics and Systems, Vol. 14, No. 2, pp. 327–339, 2004.
13. F. Miller, N. Buard, T. Carrière, R. Dufayel, R. Gaillard, P. Poirot, J. M. Palau, B. Sagnes, P. Fouillat, “Effects of Beam Spot Size on the Correlation Between Laser and Heavy Ion SEU Testing”, IEEE Transactions on Nuclear Science, Vol. 15, No. 6, pp. 3708–3715, December 2004.
14. D. Powell, J. Arlat, Y. Crouzet, “Estimators for Fault Tolerance Coverage Evaluation”, IEEE Transactions on Computers, Vol. 44, No. 2, pp. 261–274, February 1995.
15. J. Arlat, A. Costes, Y. Crouzet, J. C. Laprie, D. Powell, “Fault Injection and Dependability Evaluation of Fault-Tolerant Systems”, IEEE Transactions on Computers, Vol. 42, No. 8, pp. 913–923, August 1993.
16. H. Maderia et al. “RIFLE: a general purpose pin-level fault injector”, Proceedings of the First European Dependable Computing Conference, Berlin, Germany, October 1994, pp. 199–216.
17. P. Folkesson, S. Svensson, J. Karlsson, “A comparison of simulation based and scan chain implemented fault injection (SCIFI)”, Proceedings of FTCS-28, IEEE Computer Society Press, Munich, June 1998, pp. 284–293.
18. O. Gunnetlo, J. Karlsson, J. Tonn, “Evaluation of error detection schemes using fault injection by heavy-ion radiation”, Proceedings of the 19th Ann. Int’l Symp. Fault-Tolerant Computing, IEEE CS Press, Los Alamitos, CA, 1989, pp. 340–347.
19. 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.
20. M. C. Hsueh, T. K. Tsai, R. K. Iyer, “Fault Injection Techniques and Tools”, IEEE Computer, Vol. 30, No. 4, pp. 75–82, April 1997.
21. G. Kanawati, N. A. Kanawati, J. A. Abraham, “FERRARI: A Flexible Software-Based Fault and Error Injection System”, IEEE Transactions on Computers, Vol. 44, No. 2, pp. 248–260, February 1995.
22. J. Carreira, H. Madeira, J. G. Silva, “Xception: A Technique for the Experimental Evaluation of Dependability in Modern Computers”, IEEE Transactions on Software Engineering, Vol. 24, No. 2, pp. 125–136, February 1998.
23. T. Jarboui, J. Arlat, Y. Crouzet, K. Kanoun, T. Marteau, “Analysis of the effects of real and injected software faults: Linux as a case study”, IEEE Proceedings of 2002 Pacific Rim international Symposium on Dependable Computing (PRDC’02), 2002.
24. M. Rebaudengo, M. Sonza Reorda, “Evaluating the Fault Tolerance Capabilities of Embedded Systems via BDM”, 17th IEEE VLSI Test Symposium, pp. 452–457, Dana Point, USA, April, 1999.
25. IEEE-ISTO 5001-2003, “The Nexus Forum™ standard for a global embedded processor debug interface”, version 2.0, 2003.
26. A. V. Fidalgo, G. R. Alves, J. M. Ferreira, “Real Time Fault Injection Using Enhanced OCD – A Performance Analysis”, 21st IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT), 2006.
27. J. Peng, J. Ma, B. Hong, C. Yuan, “Validation of Fault Tolerance Mechanisms of an Onboard System”, 1st International Symposium on Systems and Control in Aerospace and Astronautics (ISSCAA), pp. 1230–1234, January 2006.
28. M. Portela-García, M. García-Valderas, C. López-Ongil, L. Entrena, “An Efficient Solution to Evaluate SEU Sensitivity in Digital Circuits with Embedded RAMs”, XXI Conference on Design of Circuits and Integrated Systems (DCIS’06), November 2006.
29. P. Kenterlis, N. Kranitis, A. Paschalis, D. Gizopoulus, M. Psarakis, “A Low-Cost SEU Fault Emulation Platform for SRAM-Based FPGAs”, 12th IEEE International On-Line Testing Symposium, pp. 235–241, July 2006.
30. M. Alderighi, F. Casini, S. D’Angelo, M. Mancini, A. Marmo, S. Pastore, G. R. Sechi, “A Tool for Injecting SEU-like Faults into the Configuration Control Mechanism of Xilinx

Uncorrected Proof

Author's Proof

852 Virtex FPGAs", 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI
853 Systems, 2003.
854 31. M. Alderighi , F. Casini, S. D'Angelo, M. Mancini, S. Pastore, G. R. Sechi, R. Weigand,
855 "Evaluation of Single Event Upset Mitigation Schemes for SRAM Based FPGAs Using the
856 FLIPPER Fault Injection Platform", 22nd IEEE International Symposium on Defect and Fault
857 Tolerance in VLSI Systems, pp. 105–113, 2007.
858 32. K. T. Cheng, S. Y. Huang, W. J. Dai, "Fault emulation: a new approach to fault grading",
859 Proceedings of the International Conference on Computer-Aided Design, pp. 681–686, 1995.
860 33. L. Antoni, R. Leveugle, B. Feher, "Using Run-Time Reconfiguration for Fault Injection in
861 HW Prototypes", IEEE Int. Symposium on Defect and Fault Tolerance in VLSI Systems,
862 pp. 245–253, 2002.
863 34. M. Aguirre, J. N. Tombs, F. Muñoz, V. Baena, A. Torralba, A. Fernandez-Leon, F. Tortosa,
864 D. Gonzalez-Gutierrez, "An FPGA based hardware emulator for the insertion and analysis of
865 Single Event Upsets in VLSI Designs", Radiation Effects on Components and Systems
866 Workshop, September 2004.
867 35. J. H. Hong, S. A. Hwang, C. W. Wu, "An FPGA-Based Hardware Emulator for Fast Fault
868 Emulation", MidWest Symposium on Circuits and Systems, 1996.
869 36. P. Civera, L. Macchiarulo, M. Rebaudengo, M. Sonza Reorda, M. Violante, "Exploiting
870 Circuit Emulation for Fast Hardness Evaluation", IEEE Transactions on Nuclear Science,
871 Vol. 48, No. 6, 2001.
872 37. C. López-Ongil, M. García-Valderas, M. Portela-García, L. Entrena, "Autonomous
873 Fault Emulation: A New FPGA-based Acceleration System for Hardness Evaluation", IEEE
874 Transactions on Nuclear Science, Vol. 54, Issue 1, Part 2, pp. 252–261, February 2007.
875 38. M. Violante, "Accurate Single-Event-Transient Analysis via Zero-Delay Logic Simulation",
876 IEEE Transactions on Nuclear Science, Vol. 50, No. 6, December 2003.
877 39. M. García Valderas, R. Fernández Cardenal, C. López Ongil, M. Portela García, L. Entrena.
878 "SET emulation under a quantized delay model", Proceedings of the 22nd IEEE International
879 Symposium on Defect and Fault-Tolerance in VLSI Systems (DFTS), pp. 68–78, September
880 2007.
881 40. F. Lima, S. Rezgui, L. Carro, R. Velazco, R. Reis, "On the use of VHDL simulation and
882 emulation to derive error rates", Proceedings of 6th Conference on Radiation and Its Effects
883 on Components and Systems (RADECS'01), Grenoble, September 2001.
884 41. F. Faure, P. Peronnard, R. Velazco, R. Ecoffet, "THESIC+, a flexible system for SEE testing",
885 Proceedings of RADECS Workshop, [September 19–20, 2002, Padova], pp. 231–234.
886 42. D. Lewis, V. Pouget, F. Beaudoin, P. Perdu, H. Lapuyade, P. Fouillat, A. Touboul, "Backside
887 Laser Testing of ICs for SET Sensitivity Evaluation", IEEE Transactions on Nuclear Science,
888 Vol. 48, Issue 6, Part 1, pp. 2193–2201, December 2001.
889 43. F. Faure, R. Velazco, P. Peronnard, "Single-Event-Upset-Like Fault Injection: A Comprehen-
890 sive Framework", IEEE Transactions on Nuclear Science, Vol. 52, Issue 6, Part 1,
891 pp. 2205–2209, December 2005.
892 44. P. Civera, L. Macchiarulo, M. Rebaudengo, M. Sonza Reorda, M. Violante, "FPGA-Based
893 Fault Injection for Microprocessor Systems", IEEE Asian Test Symposium, pp. 304–309,
894 2001.
895 45. M. Nicolaidis, "Emulation/Simulation d'un circuit logique", French patent, filed February 25
896 2005, issued October 12 2007.
897 46. M. G. Valderas, P. Peronnard, C. Lopez-Ongil, R. Ecoffet, F. Bezerra, R. Velazco, "Two
898 Complementary Approaches for Studying the Effects of SEUs on Digital Processors", IEEE
899 Transactions on Nuclear Science, Vol. 54, Issue 4, Part 2, pp. 924–928, August 2007.

Uncorrected Proof

Author's Proof

Author Queries

Chapter No.: 6

Query Refs.	Details Required	Author’s response
AU1	Please provide affiliation for all the authors.	Affiliation is written
AU2	The sentence has been edited for better readability. Please check and approve the edit.	OK
AU3	“[NEXUS]” has been changed to ref. “[25]”. Please check if this is correct.	OK
AU4	The sentence has been edited for better readability. Please check and approve the edit.	OK
AU5	The sentence has been edited for better readability. Please check and approve.	OK
AU6	The sentence has been edited for better readability. Please check and approve the edit.	OK