

# **Experimental Validation of a Fault-Tolerant System Using Physical Fault Injection**

*by*

*Peter Folkesson*

Technical Report no. 239L

Department of Computer Engineering

1996



# Experimental Validation of a Fault-Tolerant System Using Physical Fault Injection

*by*

*Peter Folkesson*



Submitted to the School of Electrical and Computer Engineering,  
Chalmers University of Technology,  
in partial fulfilment of the requirements for the degree of  
Licentiate of Engineering

Department of Computer Engineering  
Chalmers University of Technology  
S-412 96 Göteborg, Sweden

ISBN 91-7197-355-9

Göteborg, August 1996

## Abstract

This thesis describes and compares three physical fault injection techniques—heavy-ion radiation, pin-level fault injection and electromagnetic interference—and their use in the validation of the fault-tolerant, distributed, real-time system MARS. The study had two main objectives. One was to make the first direct comparison of three different physical fault injection techniques, which was possible as they were all applied on the same fault-tolerant computer system. The second objective was to make a thorough investigation of the target system to evaluate the coverage of the built-in fault tolerance features of the MARS system. The MARS system consists of computer nodes that must be *fail-silent*, i.e. if the node fails to produce a correct result, it should not produce any result at all. A *fail-silence violation* occurs if the node produces an erroneous result.

The experimental results show fairly large differences between the fault injection techniques in a study of the distribution of the error detections among the various error detection mechanisms (EDMs) built into the MARS system. This suggests that the techniques are somewhat complementary. The heavy-ion radiation technique stressed the system to the greatest extent, as indicated by the relatively large number of fail-silence violations that occurred for this technique, e.g. it was the only technique that caused any fail-silence violations when all the EDMs were activated in the MARS system. It also showed the largest spread in the detections among the EDMs. The pin-forcing technique managed to exercise the hardware EDMs located outside the CPU more effectively than the heavy-ion radiation and electromagnetic interference techniques, which instead were more effective in exercising the application level EDMs (i.e. double time redundant execution of tasks and message checksum calculations).

Each technique managed to cause many error types that the MARS designers had not anticipated. All three techniques were particularly effective in exercising EDMs implemented at the hardware level of the MARS system (all such mechanisms were exercised). However, the techniques were unable to exercise many of the mechanisms implemented by the MARS system software. In fact, most such mechanisms were never exercised by any of the techniques.

The EDMs implemented at the hardware level detected the largest amount of errors generated by the three fault injection techniques, while the mechanisms implemented at the application level detected the smallest amount of errors. Still, the application level mechanisms were shown to be necessary, and also quite efficient, for improving the fail-silence coverage.

**Keywords:** Fault Tolerance, Coverage, Experimental Evaluation, Fault Injection, Physical Techniques, Real-Time Systems, EMI, Pin-Level Fault Injection, Heavy-Ion Radiation



## Acknowledgments

Many people have contributed to the work presented in this thesis. I would first like to thank my advisors, Prof. Jan Torin and Dr. Johan Karlsson, for introducing me to an exciting research area and for their invaluable support during this work. I would then like to express my gratitude to Prof. Dr. Hermann Kopetz of the Technical University of Vienna, the chief architect of the MARS system, for his support and many valuable suggestions and comments. Special thanks go to Günther Leber, who developed the set-up used in the experiments, and also provided the results of the EMI experiments. I also thank Dr. Jean Arlat and Yves Crouzet at LAAS-CNRS for their pin-level results and inspiring views of this study. Special thanks are also due to Hans Bergstrand and Lennart Hansson for their technical assistance with the experimental set-up.

This work was partially supported by PDCS2 (Predictable Dependable Computing Systems 2), Basic Research Project No. 6362 of the ESPRIT programme.





---

# Table of Contents

<b>1 Introduction</b> .....	1
<b>2 The MARS System</b> .....	3
2.1 MARS System Introduction .....	3
2.2 Timeliness .....	3
2.3 MARS Hardware Architecture .....	4
2.4 MARS Software Architecture .....	5
2.5 Fault Tolerance Features .....	7
2.5.1 Level 1: Hardware EDMs .....	8
2.5.2 Level 2: System Software EDMs .....	9
2.5.3 Level 3: Application Level EDMs .....	9
<b>3 Three Physical Fault Injection Techniques</b> .....	10
3.1 Heavy-Ion Radiation .....	10
3.2 Pin-Level Injection .....	11
3.3 Electromagnetic Interference .....	12
3.4 Comparison of the Techniques .....	13
3.4.1 Controllability Space/Time .....	14
3.4.2 Flexibility .....	14
3.4.3 Reproducibility .....	15
3.4.4 Physical Reachability .....	16
3.4.5 Timing Measurement .....	16
<b>4 Common Experimental Set-up</b> .....	17
4.1 Test Application Used .....	17
4.2 Hardware Configuration .....	18
4.3 Detailed Operation of the Set-up .....	19
4.4 Measurements .....	20
4.4.1 Experimental Assessment .....	20
4.4.2 Predicates .....	21
<b>5 Experimental Results</b> .....	24
5.1 Experimental Combinations Used .....	24
5.2 Format of Presented Results .....	25
5.3 Results of Heavy-Ion Radiation .....	26
5.4 Results of Pin-Level Injection .....	30
5.5 Results of EMI .....	33
<b>6 Detailed Analysis of Experimental Results</b> .....	35
6.1 Comparison of Results .....	35
6.2 Detailed Comparison .....	37
6.2.1 Distribution of Errors Among CPU EDMs .....	37
6.2.2 Distribution of Errors Among NMI EDMs .....	38
6.2.3 Distribution of Errors Among System Software EDMs .....	41
6.2.4 Distribution of Errors Among Other Mechanisms .....	42
6.3 Fail-Silence Violations Observed .....	43



Table of Contents

---

6.4 Error Detection Mechanisms Exercised .....	45
<b>7 Conclusions</b> .....	48
<b>References</b> .....	51
<b>Appendix A: Acronyms used</b> .....	54



# 1 Introduction

The area of *fault-tolerant computing* is becoming increasingly more important as computers gain popularity in safety-critical applications in the vehicle, aeronautics, military and aerospace industries.

[Laprie 1985] established the following fundamental taxonomy in this area: A system *failure* occurs when the service provided by a system deviates from the specified service. The failure occurs because an *error* is present in the system (e.g. some unit providing an incorrect value). The phenomenological cause of that error (e.g., a programmer's mistake, a physical line stuck at ground potential or an electromagnetic perturbation) is called a *fault*.

A system that can continue to provide its specified service even when it is affected by a fault is called a *fault-tolerant system*, i.e. when the fault causes an error in the system, it is detected and some action is taken to prevent a system failure from occurring.

To evaluate fault-tolerant computer systems, they can be subjected to *fault injection*, which is a way of accelerating the occurrences of faults so that the fault tolerance mechanisms of the system can be thoroughly tested. There are several ways to inject faults into computer systems depending on *when* in the development process of the system the fault injection is made and *where* in the system it is made [Iyer and Tang 1993]. There are two fundamental techniques used for fault injection: *Software simulations* made in the design phase and *physical fault injection* made on prototypes. Physical fault injection therefore injects faults into a real hardware version of the target system, while software simulations use a software-implemented model of the target system.

Physical fault injection can be divided into (i) *hardware-implemented fault injection*, made with the aid of additional hardware in the system, and (ii) *software-implemented fault injection (SWIFI)*, made by introducing faults into the contents of memory or registers or by emulation of hardware and software faults. The techniques examined in this thesis all belong to the first category. Although SWIFI has many advantages, e.g. it has higher flexibility and lower cost compared with the other techniques, the correspondence between the faults implemented by software and the actual hardware faults occurring in the system has not yet been established with confidence.

The target system for the fault injection experiments conducted in this study is MARS (the MAintainable Real-time System), a time-triggered, fault-tolerant, distributed computer system developed at the Technical University of Vienna [Kopetz *et al.* 1989]. It consists of several computer nodes communicating by means of a synchronous time division multiple access (TDMA) strategy. The nodes contain extra hardware and software for fault tolerance and can be configured to operate in redundancy, i.e. when two nodes execute the same tasks.

This study had two main objectives. One was to make the first direct comparison of three physical fault injection techniques, which was possible because they were all applied on the same fault-tolerant computer system [Karlsson *et al.* 1995]. Thus it was also possible

to investigate the usefulness of the techniques and whether or not they are complementary, i.e. whether or not they exercise different error detection mechanisms. This also led to a very thorough investigation of the MARS system. The second objective was therefore to evaluate the coverage of the built-in fault tolerance features of the MARS system.

The three physical fault injection techniques used in the experiments were: *heavy-ion radiation*, used at Chalmers University of Technology, Göteborg, Sweden; *pin-level fault injection*, used at LAAS-CNRS, Toulouse, France; and *electromagnetic interference*, used at the Technical University of Vienna, Vienna, Austria. The study, which was launched within the framework of the PDCS-2 project of the ESPRIT program (Basic Research Project No. 6362), therefore engaged these three sites, and a common experimental set-up using five MARS nodes was implemented at each site to conduct a coherent set of experiments.

Chapter 2 gives an overview of the MARS system, the hardware that is used for operation and for error detection, and how the operating system and the application software is constructed and executed. Chapter 3 gives a description and a general comparison of the three physical fault injection techniques used for validating the MARS system. The common experimental set-up used at the three sites is described in Chapter 4, and the results obtained using this set-up are given in Chapter 5. In Chapter 6, the results are analysed and compared in greater detail. Finally, the conclusions of this study are given in Chapter 7.

## 2 The MARS System

This chapter gives an overview of the target system MARS (MAintainable Real-time System), which was used for the fault injection experiments conducted in this study. The hardware, software and fault tolerance features built into the system are described.

### 2.1 MARS System Introduction

One approach toward achieving fault-tolerant distributed real-time systems is to use a number of autonomous, *fail-silent* processing nodes that are interconnected by a real-time network and communicate by exchanging messages [Powell *et al.* 1988]. The MARS system has been realized on the basis of this approach. The fundamental fault tolerance property of each processing node in the MARS system is therefore to be fail-silent, i.e. the node is shut down when an error is detected within it to avoid error propagation. If the node would continue to operate despite an error, the error may result in an erroneous message, i.e. a *fail-silence violation*, which may further lead to a system failure when the message is used by the application (e.g. controlling a rolling mill). The advantage of using fail-silent nodes rather than nodes that always try to produce correct results lies in their much simpler design and therefore more cost-effective manufacturing.

There are two types of fail-silence violations that can occur in the MARS system: One occurs when the message produced by the node is erroneous, a fail-silence violation in the *value* domain; and another when a node sends a message (correct or not) at an illegal point in time (when a different message from another node is expected), a fail-silence violation in the *time* domain.

### 2.2 Timeliness

The issue of timeliness is always important in discussions of real-time systems. All relevant actions in the MARS system must be scheduled before operation, i.e. the MARS system is strictly *time triggered* in order to guarantee proper timing behaviour.

The following main actions are considered [Reisinger *et al.* 1995]:

- the points in time at which a processing node may send a message (a Time Division Multiple Access (TDMA) protocol is used) and the types of messages that may be sent at specific points in time,
- the start times and deadlines of all processes,
- the points in time when sensor values are read and actuator values are written, and
- the processing steps for recovery and integration of failed nodes (for preventing a properly detected fault from affecting the correct timing behaviour of the system).

Thus the system designer needs to determine the following:

- the maximum execution time of each process,
- the maximum time for communication,
- the operating system overhead, and
- the overhead of hardware activities that influence the timing behaviour of the node.

The precise global time base needed by every time triggered system in order to synchronize the actions within different processing nodes of the system is maintained by a distributed, fault-tolerant clock synchronization algorithm described in [Kopetz and Ochsenreiter 1987].

### 2.3 MARS Hardware Architecture

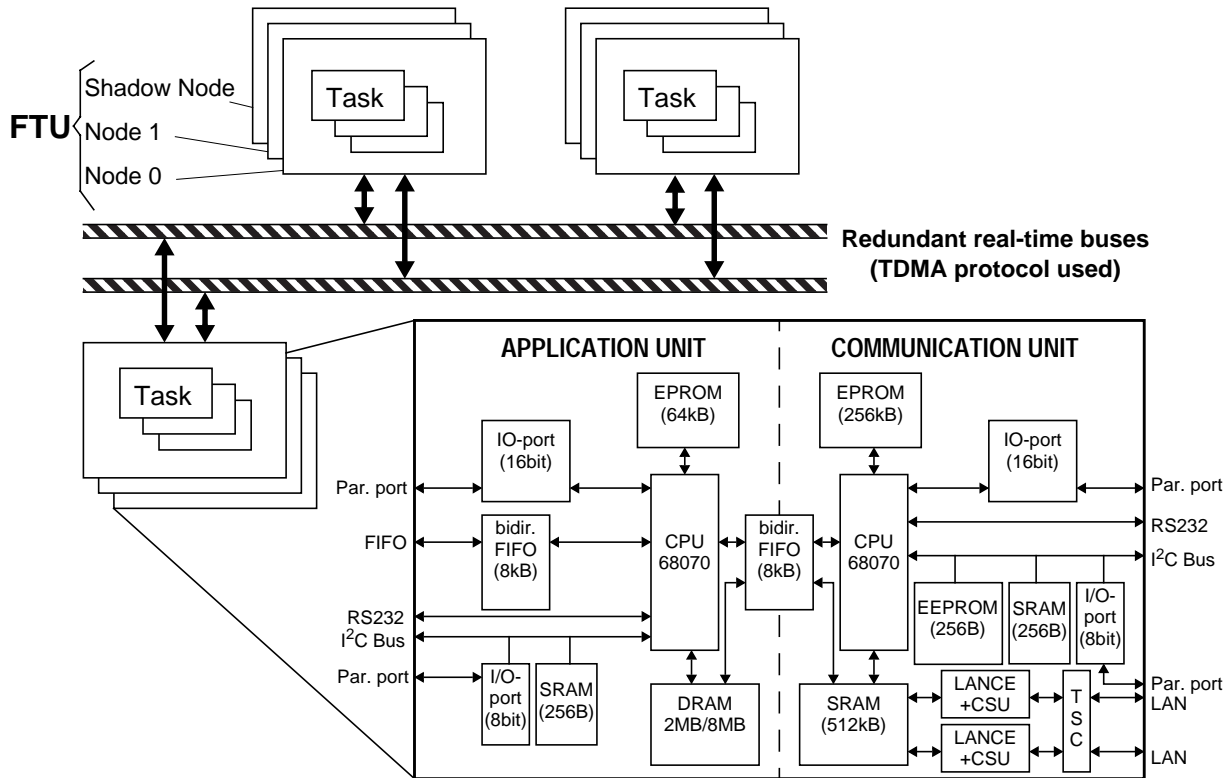
The MARS system consists of independent computer units called FTUs (Fault-Tolerant Units) that communicate on a redundant real-time bus, known as the MARS bus. These FTUs consist of up to three autonomous processing nodes known as SRUs (Smallest Replaceable Units), hereafter called *nodes*. Various error detection mechanisms are incorporated into the nodes for achieving fail-silence. In order to form an FTU, two nodes can be configured to run in active redundancy together with an optional third node running as a stand-by shadow node, i.e. not active on the MARS bus, but still executing the same software as the other two nodes (see Figure 1). When an error is detected in an active node, this node is shut down and the shadow node is made active on the bus, thus restoring the initial degree of redundancy. The MARS system is therefore said to have a two-layered approach for achieving fault tolerance, i.e. a bottom layer responsible for error detection and error confinement (node shut down) and a top layer providing enough redundancy to be able to tolerate silent failures of parts of the system (handling redundant data and reconfiguration of the system in case of a node failure) [Grünsteidl and Kopetz 1991].

Communication between FTUs is carried out on two redundant real-time buses using a TDMA-based protocol [Grünsteidl and Kopetz 1991; Grünsteidl, Kantz and Kopetz 1991]. Each FTU is given a certain time slot in which to communicate with the other FTUs in the system, further divided into SRU time slots for each active node of the FTU. During the first SRU time slot, Node 0 sends a message on the first real-time bus and Node 1 sends a message on the second real-time bus. During the next SRU time slot, the transmission channels are switched so that Node 0 now sends the same message on the second real-time bus, while Node 1 sends on the first real-time bus. Due to the fail-silence property of one node, all four redundant messages sent are assumed to be correct and may be used interchangeably by the other nodes in the system.

The structure of each node is also given in Figure 1 and described in detail in [Steininger and Reisinger 1993]. The node consists of two self-contained computer units, the *application unit*, executing the application dedicated to the system, and the *communication unit*, dealing with communication with other nodes, clock synchronization, membership service etc. The only way for the two units of the node to communicate is via a bidirectional FIFO.

---





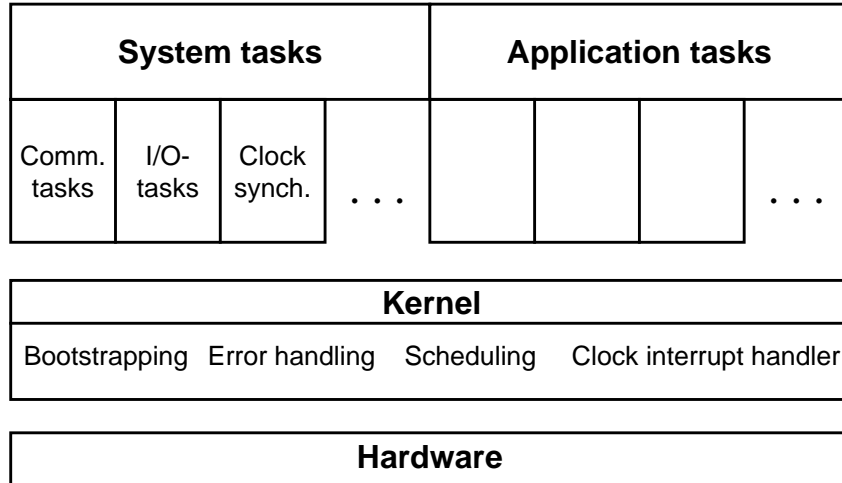
**Figure 1: The MARS hardware**

Each unit is built around a Philips 68070 CPU (clocked with a frequency of 15 MHz), which is a Motorola 68000-based CPU that includes a MMU, two-channel DMA controller, UART (RS232) interface, Inter-IC ( $I^2C$ ) bus and an interrupt controller [Philips Semiconductors 1991]. Both units have their own memory, ports and additional hardware for error detection (for preventing fail-silence violations in the value domain). The communication unit is also equipped with two LANCE (Ethernet) chips (Am7990) for communication on the MARS bus, together with two Clock Synchronization Units (CSU) for maintaining a global time base. Additionally, a Time Slice Controller (TSC) that supervises access to the MARS bus for preventing faulty nodes from disturbing non-faulty ones (thus preventing fail-silence violations in the time domain) and a watchdog timer are built into the communication unit.

## 2.4 MARS Software Architecture

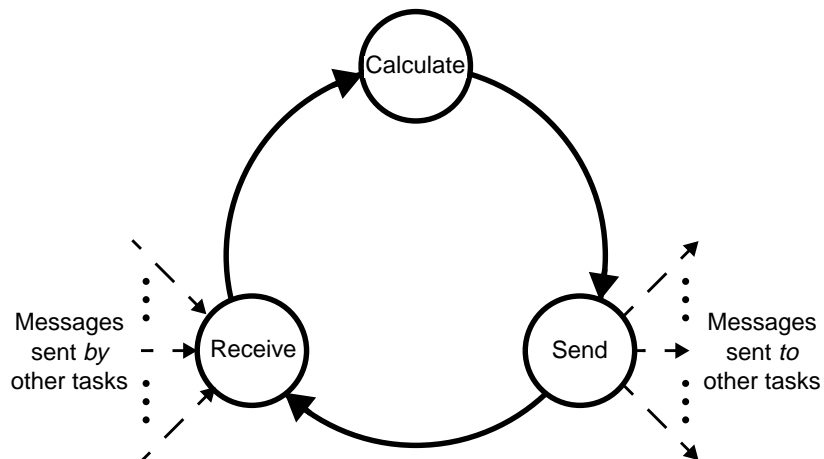
The operating system software in MARS is based on a microkernel architecture [Reisinger 1993] (see Figure 2). The kernel deals only with bootstrapping, error handling, scheduling according to pre-calculated dispatching tables and clock interrupt handling solely used for dispatching purposes.

The remaining parts of the operating system are implemented as tasks at the application level. These tasks deal with communication, I/O, clock synchronization etc.



**Figure 2: The MARS software**

All tasks execute in a periodic receive-calculate-send loop [Krüger and Nossal 1993] (see Figure 3). They first enter a receive phase where data is received from other tasks, the data is then processed in a calculate phase and, finally, the resulting data are sent to other tasks in a send phase. This periodic receiving and sending of messages has several advantages, e.g. the maximum execution time of the task is easier to calculate and the concept of time redundant execution (see Section 2.5.3) is easier to implement. Tasks are often related and can therefore be grouped together to form *teams*, which are able to use and manipulate the same data. The data (also known as *messages*) are either stored in a global area shared by all tasks in a team or sent by various system tasks to other tasks executing anywhere in the MARS system (i.e. even to other nodes).



**Figure 3: Periodic receive-calculate-send loop for MARS tasks**

All scheduling is done using pre-calculated dispatching tables known as Application Definition Files (ADFs) in order to guarantee the proper timing behaviour of the system. A schedule consists of a list of *chains* containing lists of *threads* (tasks) that are to be executed sequentially, i.e. there is no preemption. Chains may preempt each other, however, in order to allow very short threads (with execution times shorter than the period of the clock interrupt) to execute without degrading system performance. The time for executing the whole list of chains (i.e. the time for executing one receive-calculate-send sequence of all tasks in the MARS system) is known as the *application cycle*.

## 2.5 Fault Tolerance Features

Several Error Detection Mechanisms (EDMs) are built into each MARS node in order to fulfil the fail-silence property. They are categorized into three levels according to where in the system architecture they operate (i.e., hardware level, system software level, or application level) (see Figure 4).

Immediately upon the detection of an error by any of these mechanisms, an exception handler, which is mapped to the mechanism that detected the error, is executed. The exception handler collects information about the mechanism together with the status of the CPU (register contents, MMU setup, current address used etc.) and stores this in a non-volatile SRAM connected to the I<sup>2</sup>C bus. The input of the watchdog timer is triggered for the last time at the start of the exception handler routine, causing the node to be reset by the watchdog timer after 280 ms. After reset, the stored error report is transmitted on the serial RS232 ports of the node to make later analysis possible.




	<b>LEVEL 3</b> <b>Application Level</b>	<b>End-to-end checksums</b> <b>Double execution of tasks</b>
	<b>LEVEL 2</b> <b>System Software</b>	<b>Compiler:</b> <i>Value range overflow, Loop iteration bound overflow</i> <b>OS:</b> <i>Processing time overflow, Various checks on OS data, Various assertions in the OS</i>
	<b>LEVEL 1</b> <b>Hardware</b>	<b>CPU:</b> <i>Bus error, Address error, Illegal opcode, Privilege violation, Division by zero, Stack format error, Uninitialized vector interrupt, Spurious interrupt</i> <b>Additional hardware (NMI):</b> <i>Power failure, Parity error, FIFO over/underflow, Memory fault, Illegal access to MARS bus, Error of an external device, Error of the other unit</i> <b>Watchdog timer</b>

Figure 4: MARS error detection mechanisms

### 2.5.1 Level 1: Hardware EDMs

At the bottom level are the hardware EDMs, i.e. mechanisms that are implemented in the CPU or by using extra hardware in the node.

Built into the CPU are the following EDMs:

- Bus error: Activated by the on-chip MMU when access to an unallowed address is made.
- Address error: Activated when access to a misaligned address occurs (e.g. word data is accessed on an odd address).
- Illegal opcode: Activated when an invalid instruction is read.
- Privilege violation: Activated when a privileged instruction is executed in user mode.
- Division by zero: Activated when division by zero occurs.
- Stack format error: Activated when the contents of the stack is incorrect upon return from exception processing.
- Uninitialized vector interrupt: Activated when the vector number could not be provided by the interrupting device.
- Spurious interrupt: Activated when the interrupting device does not respond to interrupt acknowledge.

The following EDMs are implemented using extra hardware that generates a non-maskable interrupt (NMI) when an error is detected. Consequently, they are called NMI EDMs:

- Power failure: Activated by an IC when the supply voltage falls below a certain threshold value.
- Parity error: Parity bits are calculated on data transferred to memory (DRAM or FIFO). During transfer from memory, the parity bits are checked and any errors are reported.
- FIFO over/underflow: Activated when the FIFO memory becomes flooded or when data is being read from an empty FIFO.
- Memory fault: Activated by the address decoder logic when access to physically non-existing memory occurs.
- Illegal access to MARS bus: Activated by the TSC when the node tries to send outside its intended time slots.
- Error in external device: Provided for external devices connected to the node.
- Error in other unit: Activated by one unit of the node when an error occurs in the other unit.

The error signals originating from all of these mechanisms are connected to one input latch each. The NMI signal provided for the CPU is simply the logical OR of all values of these input latches, and the system may know which NMI mechanisms (one or several at

---

the same time) have been triggered by reading the contents of the input latches.

The extra hardware also consists of the watchdog timer that is built into the communication unit. It activates a node reset when the watchdog input remains unaffected for more than 280 ms.

In addition to the mechanisms provided by the CPU and those provided by extra hardware, faults can also trigger unexpected exceptions. These are exceptions to which neither the EDMs built into the CPU nor the mechanisms implemented using extra hardware were mapped, but the corresponding exception handlers were nevertheless provided by the system for increased robustness.

### 2.5.2 Level 2: System Software EDMs

At the next level are the EDMs that are implemented in the operating system software or generated by the compiler for the application.

EDMs generated by the compiler (Compiler Generated Run-Time Assertions, CGRTA):

- Value range overflow: Activated when arithmetic overflow occurs or variable range checks fail.
- Loop iteration bound overflow: Activated when loop variables have values outside their specified bounds.

EDMs built into the operating system:

- Processing time overflow: Activated when a task has not finished before its deadline.
- Various checks on OS data: Integrity checks on operating system data that fail.
- Various assertions in the OS: Assertions coded into the operating system that fail.

### 2.5.3 Level 3: Application Level EDMs

At the highest level are the so called application level EDMs used for implementing the *extended* fail-silence property of the node (i.e. the node is considered fail-silent even when a detectably corrupt message is being sent) or for detecting errors caused by transient faults:

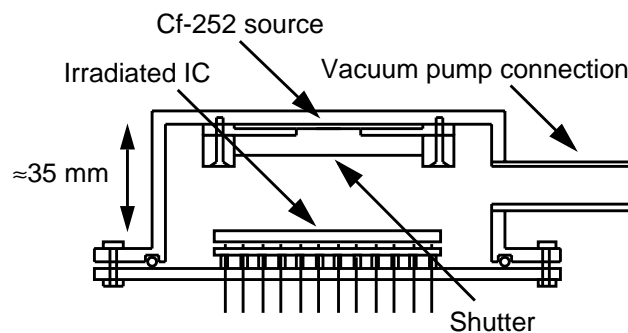
- End-to-end checksums: Checksums (16 bit CRC) that are calculated upon all messages sent between tasks. Detection of a corrupt message activates this EDM. Thus, the node is considered fail-silent even when a detectably corrupt message has been sent.
- Double execution of tasks: All tasks are executed in time redundancy using duplicated text and data images, and the messages produced by the two executions are compared for detecting errors caused by transient faults.

## 3 Three Physical Fault Injection Techniques

This chapter describes the three physical fault injection techniques used in the validation of the MARS system. The heavy-ion radiation and pin-level techniques are well known techniques that have been used in several studies in the past, while the EMI technique has never been used for evaluating fault tolerance mechanisms before. A comparison of the main features of the techniques, in the way that they were applied in this study, is also given.

### 3.1 Heavy-Ion Radiation

The heavy-ion radiation technique is comprehensively described in [Karlsson *et al.* 1994]. The technique is based on the fact that heavy ions emitted from a Cf-252 source may cause bit-flips (also known as Single Event Upsets, SEUs) in internal locations in integrated circuits. To use this technique, the packaging containing the IC must be opened and the IC placed in a vacuum together with the Cf-252 source. This is necessary as the heavy ions are attenuated by air molecules and other materials. A miniature vacuum chamber was developed which contains the Cf-252 source and the irradiated IC (see Figure 5). The IC's pin connections are extended through the bottom plate of the miniature vacuum chamber, so that the chamber can be plugged directly into the socket of the irradiated circuit in the tested system. An electrically manoeuvred shutter is used for controlling the time at which fault injection may occur by shutting the radiation on and off.



**Figure 5: Miniature vacuum chamber cross-sectional view**

There are other ways of creating SEUs for circuit testing using cyclotrons [Sokol *et al.* 1987; Elder *et al.* 1988]. This is obviously a more expensive technique, and it is debatable whether controllability is better for this technique (i.e. whether the heavy ions can be more easily directed to specific parts of the ICs in this technique). Some controllability has been shown to be possible for the heavy-ion radiation technique using shielding [Lidén *et al.* 1994].

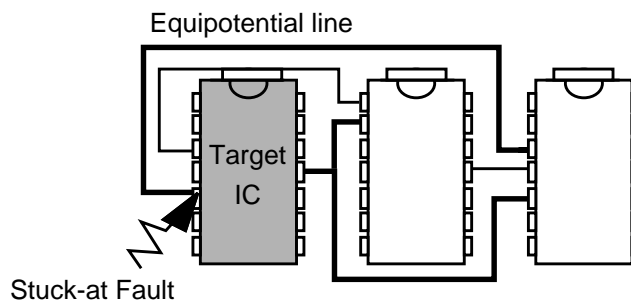
A commercially available Cf-252 source with a nominal activity of 15 kBq was used in

the heavy-ion radiation experiments. About three percent of all disintegrations generate fission fragments (heavy ions) and the rest generate alpha particles. The alpha particles do not affect most circuits because they deposit much less charge in silicon than do the fission fragments (linear energy transfer of about  $1.6 \text{ MeV mg}^{-1} \text{ cm}^2$  vs.  $41\text{-}45 \text{ MeV mg}^{-1} \text{ cm}^2$  for most fission fragments) [Gunnflo *et al.* 1987].

In CMOS circuits, the heavy-ion radiation may cause latch-ups, i.e. the triggering of a parasitic four-layer switch (npnp or pnpn) that acts as a silicon-controlled rectifier. The latch-up results in excessive heat dissipation, which may destroy the circuit and is indicated by a drastic increase in the current drawn by the circuit. A special device acting as a current guard was therefore developed to protect the circuit by shutting off the power when the current exceeds a threshold value.

### 3.2 Pin-Level Injection

Pin-level fault injection is the most widely used physical fault injection technique. It has been used for validating fault-tolerant distributed computer systems (e.g., [Damm 1986; Walter 1990]), and for evaluating the coverage of specific mechanisms such as error detection by means of signature analysis [Schuette *et al.* 1986].



**Figure 6: Pin-level fault injection using forcing**

In this technique, faults are injected on the pins of the ICs of the tested system. A variety of fault models is used; e.g. *stuck-at* 0 or 1, in which the faulted pins are set to a logic 0 or 1, *bridging*, when several pins of a circuit are interconnected, *inverted signal*, in which the faulted pin level is inverted, or *open connection*, when the faulted pin is essentially tri-stated. The duration of the fault may vary to simulate transient, intermittent and permanent faults. There are two main implementations of this technique:

- *Forcing*: Faults are injected by multi-pin probes that are directly applied on the pins of the ICs and associated equipotential lines (see Figure 6).
- *Insertion*: The target ICs are removed from the system and plugged into a separate box where transistor switches connected to the pins of the ICs ensure proper isolation from the rest of the system. All associated equipotential lines will therefore remain unaffected for each input pin of the target IC, as opposed to the forcing technique where all circuits

connected to the faulted input pin are affected.

Several tools for pin-level fault injection have been developed, e.g. the tool used for FTMP [Lala 1983], MESSALINE [Arlat *et al.* 1990], or RIFLE [Madeira *et al.* 1994]. The tool used in this study is MESSALINE (see Figure 7), developed at LAAS-CNRS. It supports both implementations above, but only the forcing technique was used in this study. The fault models supported in addition to stuck-at (0 or 1) are open connection and inverted signal in the case of insertion, and bridging in the case of forcing. The duration of the fault is fully controllable, as is the point in time at which the fault is activated as it is possible to use the signals of the target system as fault triggers.

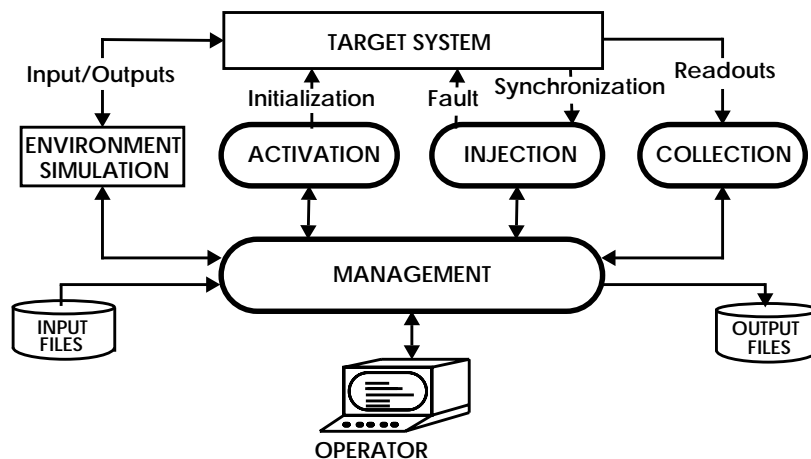


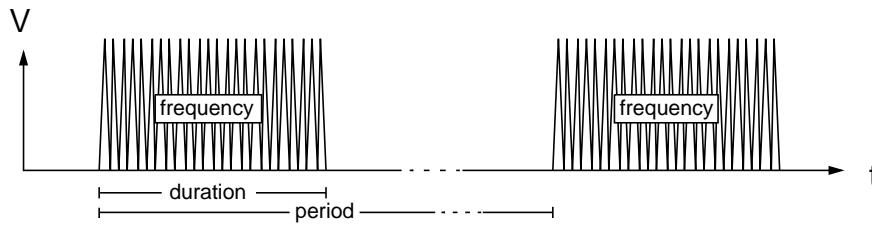
Figure 7: General architecture of MESSALINE

### 3.3 Electromagnetic Interference

Electromagnetic interference (EMI) may occur in motorized vehicles and industrial plants, causing computer failures when computers are put to use in such environments. The use of EMI for evaluating the MARS system was therefore investigated in this study.

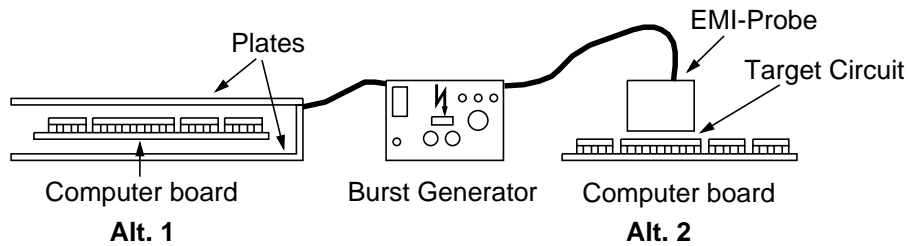
The EMI disturbances used in these experiments were produced by a burst generator which generated bursts conforming to the IEC 801-4 standard (CEI/IEC), i.e. bursts with a duration of 15 ms, a period of 300 ms, a frequency of 1.25, 2.5, 5 or 10 kHz, and voltage levels between 225 V to 4400 V (see Figure 8). These bursts are similar to those which arise when switching inductive loads with relays or mechanical circuit breakers.





**Figure 8: Electromagnetic bursts**

There were two ways of injecting faults using EMI in this study (see Figure 9). Alternative 1 used two conducting plates connected to the burst generator as a fault injector. The whole computer board of the fault-injected MARS node was placed between the plates, and small pieces of wire acting as antennas were connected to the pins of the target ICs on the computer board. Alternative 2 used a special probe that was placed above the target ICs in order to focus the EMI bursts to them. The probe was used both with and without the antennas connected to the pins of the target ICs.



**Figure 9: EMI fault injection set-ups**

### 3.4 Comparison of the Techniques

The three physical fault injection techniques used in this study can be compared according to five attributes: *Controllability*, with respect to both *space* and *time*, *flexibility*, *repeatability*, *physical reachability* and *timing measurement*. Table 1 gives a characterization of the techniques according to the five attributes graded on the scale *none*, *low*, *medium* and *high*.

Attributes	Heavy-ion	Pin-level	EMI
Controllability, space	low	high	low
Controllability, time	none	high/medium	low
Flexibility	low	medium	high/medium
Reproducibility	medium	high	low
Physical reachability	high	medium	medium
Timing measurement	medium	high	low

**Table 1: Characterization of the fault injection techniques**

### 3.4.1 Controllability Space/Time

The controllability attribute is divided into a space domain and a time domain. The space domain corresponds to controlling *where* the faults are injected, while the time domain corresponds to controlling *when* the faults are injected.

#### Heavy-ion radiation

The controllability in the space domain in the heavy-ion technique is low. It is possible to make a selection of which circuits to irradiate and even to use shielding for a rough selection of the parts of the circuit that should be irradiated (shielding was not used in this study, however), but the exact locations at which to inject faults can not be chosen. Controllability in the time domain is impossible since the faults are generated by heavy ions created by a random decay process of the Cf-252 source (although the shutter mechanism described in Section 3.1 may give some control of when *no* fault injection may occur).

#### Pin-level fault injection

Pin-level fault injection has high controllability in both the space and time domains. The exact pin locations to fault-inject are easy to find, and timing controllability may only be difficult when the clock frequency of the target circuit is high and one wishes to synchronize fault injection with the activity of the circuit.

#### EMI fault injection

EMI fault injection has low controllability in the space domain, both when the probe and the plates are used. The probe may affect circuits surrounding the target circuit, and the plates affect the whole computer board, although antennas may give some controllability as to the circuits the fault injection should be focused. While the time of injection can be synchronized with the system activity, it is difficult to determine exactly when a fault is injected, and thus the controllability in time is low as well.

### 3.4.2 Flexibility

The flexibility attribute expresses the effort needed to modify the experimental set-up so

that another target circuit is used (if at all possible).

### **Heavy-ion radiation**

The flexibility of heavy-ion radiation is low since considerable preparations are necessary for changing target circuit. The circuit package must be opened and mechanical and electrical adaptation between the target system and miniature vacuum chamber carried out, and, in the case of latency measurements, the development of a comparator card is needed. Typically, only a few highly integrated key components of the system are therefore fault injected using heavy-ion radiation.

### **Pin-level fault injection**

The flexibility of the pin-level fault injection technique is higher than for the heavy-ion technique, especially when a fault injection tool such as MESSALINE is used. Extra load capacitances introduced by the connection probes may interfere with the target system and the pins of some modern ICs may be difficult to access physically. Flexibility is therefore considered medium.

### **EMI fault injection**

EMI has a high flexibility when no antenna wires are used, as there is no physical connection between the target circuit and the fault injector. Connection of antenna wires to the pins of the target circuit faces the same accessibility problems as the pin-level technique, and flexibility in that case is only medium.

## **3.4.3 Reproducibility**

Reproducibility refers to the ability to do two things:

- *Statistically reproduce the results when using a certain set-up* in order to ensure the credibility of the fault injection experiments when used in validating a system.
- *Repeat individual fault injections exactly*, which is needed when the aim of the experiments is to remove faults in the design of the fault tolerance mechanisms.

### **Heavy-ion radiation**

Previous research has shown that results of heavy-ion radiation experiments are statistically reproducible among different samples of target ICs. Exact repeatability is impossible due to the lack of controllability. Reproducibility is therefore considered low.

### **Pin-level fault injection**

With the pin-level fault injection tool MESSALINE, it is possible to both statistically reproduce the results and repeat individual fault injections exactly. Reproducibility is therefore high in pin-level fault injection.

### **EMI fault injection**

Small changes in the positioning of the probe or antenna wires in the EMI fault injection technique may affect statistical reproducibility. The results of the EMI experiments (see

Section 5.5) show that reproducibility is difficult to achieve. Exact repeatability is not possible for EMI. The reproducibility is therefore low.

### **3.4.4 Physical Reachability**

Physical reachability expresses the ability to reach possible fault locations in a system.

#### **Heavy-ion radiation**

Heavy-ion radiation has a high physical reachability as the faults are injected in internal locations of the circuits.

#### **Pin-level fault injection**

The physical reachability in pin-level fault injection varies depending on the level of integration of the target system. Physical reachability is low for systems consisting of only a few VLSI circuits. For less integrated systems such as the MARS system, which uses a mixture of VLSI, LSI, MSI and SSI circuits, physical reachability can be considered medium.

#### **EMI fault injection**

EMI fault injection demonstrates a physical reachability similar to that of pin-level injection as most faults are injected via digital input/output signals. However, faults may also occur internally in ICs as a result of disturbances propagated through the power supply lines.

### **3.4.5 Timing Measurement**

Timing measurement expresses the ability to acquire timing information about monitored events, such as measurements of error detection latency.

#### **Heavy-ion radiation**

Measurement of error detection latency using heavy-ion radiation requires the use of the *golden chip technique*, in which the target IC must operate synchronously with a reference IC that is not subjected to fault injection to detect fault occurrences. This may be hard to achieve for certain ICs, e.g. when a varying number of wait state cycles are inserted during memory accesses. Timing measurement capability is therefore medium in heavy-ion radiation.

#### **Pin-level fault injection**

Timing measurement capability is high in the pin-level fault injection technique, as the time of injection is explicitly known.

#### **EMI fault injection**

EMI fault injection has a low ability to acquire timing information even when the golden chip technique is used, as it is difficult to confine the disturbances to the target circuit only.

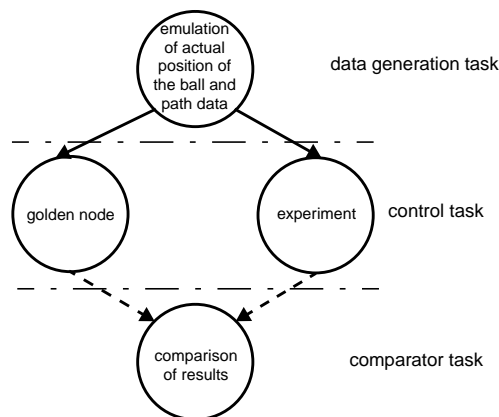
## 4 Common Experimental Set-up

The fault injection experiments were conducted at three sites: Heavy-ion radiation using Cf-252 at Chalmers University of Technology, Göteborg, Sweden; pin-level fault injection using the MESSALINE tool with forcing at LAAS-CNRS, Toulouse, France; and electromagnetic interference using conducting plates or probe, at the Technical University of Vienna, Vienna, Austria.

This chapter describes the operation of the common experimental set-up used at each site (similar to the one used in [Damm 1988]) for conducting a coherent set of experiments and the method applied for measuring the coverage of the built-in fault tolerance features of the MARS system.

### 4.1 Test Application Used

The test application used in the experiments is based on the rolling ball experiment [Kopetz *et al.* 1991], in which a ball rolls along a circular path on a plane with the help of servo motors controlling the two horizontal axes of the plane. By observing the ball with a video camera it is possible to correct for external disturbances of the position of the ball. Thus, it is a typical real-time application, chosen in order to obtain a fairly realistic workload. This is important since the error detection coverage is highly dependent on the activity of the system.



**Figure 10: Tasks and message flow**

The rolling ball experiment is completely emulated using software that is written in Modula/R, a programming language based on Modula-2 for writing real-time applications supporting the MARS architecture [Vrchticky 1992]. The application consists of three tasks (see also Figure 10):

- 1) The *data generation task*, which acts as the video camera and continuously generates

data about the nominal and actual values of the position, speed and acceleration of the ball for the control task.

2) The *control task*, which ‘controls’ the two horizontal axes of the plane. This continuously receives the emulated data from the data generation task and performs some calculations on these data, i.e. it calculates the desired acceleration of the ball. The control task does not preserve any data or state information between its periodic executions and is run in active redundancy by two MARS nodes in this experimental set-up.

3) The *comparator task*, which receives the results delivered by the two MARS nodes that run the control task in active redundancy and compares them. It delivers information about any discrepancies between the results (i.e. fail-silence violations), together with status information about the experiment, such as the number of application cycles that have been executed (the application cycle period is 40 ms) and the number of errors encountered. It also controls the fault injection device and the power supply of the tested node.

## 4.2 Hardware Configuration

The hardware configuration used in the experiments is shown in Figure 11. Five MARS nodes and a UNIX workstation were needed. As the compiler environment for the Modula/R programming language is available only on UNIX workstations, the application (together with most of the operating system) had to be downloaded from the workstation to the MARS system using a MARS node configured as a gateway between the Department’s regular Ethernet network and the MARS bus (Node 0 in Figure 11). The workstation was also responsible for collecting data from the experiments running a program called ‘observe’ (see Section 4.4.1).

The data generation node (Node 2) executed the data generation task which provided data for the control task running on FTU 1. FTU 1 was configured using two MARS nodes, one which was subjected to physical fault injection, called the tested node, and one acting as a golden node (i.e. always delivering correct data). A fifth node (Node 3) executed the comparator task, i.e. was used for checking the output of the two nodes running the control task, delivering status information to the workstation and for controlling the fault injection device and power supply of the tested node.

To give an indication of the complexity of the workload used in the set-up, the size of the code executed by the operating system and the application in both units on each MARS node are shown in Table 2.

The fault injection device consisted of either the miniature vacuum chamber for the heavy-ion radiation technique (HI), an injection probe directly connected to the pins of the target ICs for pin-level fault injection using pin-forcing (PF) or the probe or plates for the EMI technique (see Chapter 3).

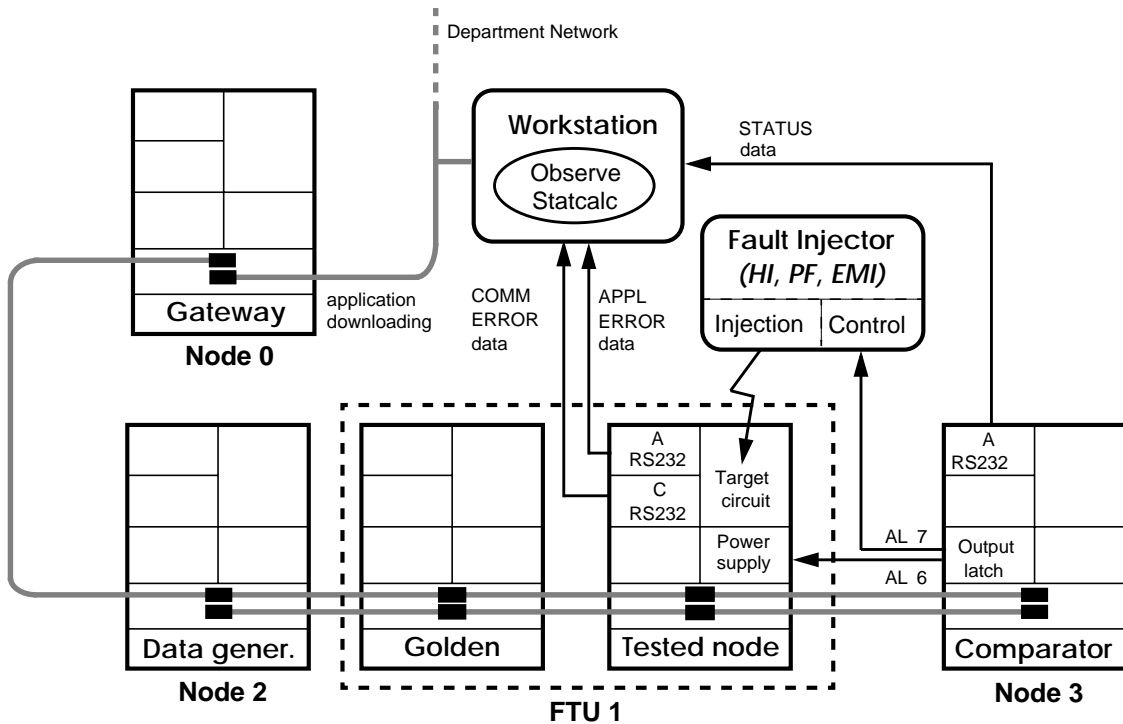


Figure 11: Hardware configuration of the experimental set-up

MARS node	Application unit: Operating system code	Application unit: Application code	Communication unit: Operating system code	Communication unit: Application code
Gateway	162 KB	0 KB	523 KB	0 KB
Data generation	162 KB	80 KB	435 KB	0 KB
Golden/Tested	162 KB	57 KB	435 KB	0 KB
Comparator	162 KB	96 KB	435 KB	0 KB

Table 2: Size of code executed at each MARS node

### 4.3 Detailed Operation of the Set-up

Faults were injected into the tested node until the comparator node detected a failure (i.e. a discrepancy in the data delivered by the tested node and golden node was observed) or when a resulting error was detected by the tested node, causing it to store an error report (i.e. data about which error detection mechanism detected the error, together with the state of the CPU when the error was detected, e.g. the contents of the CPU registers) in non-volatile SRAM in each unit and then reset itself. After reset, both units of the tested node delivered their stored error reports via serial RS232 ports to the workstation for further

analysis. No error information was ever sent on the serial ports of the tested node if the error was detected by the comparator node alone. The workstation then instead retrieved the necessary error information from the status data delivered by the comparator node.

The comparator node then shut off the power to the tested node via the signal AL 6, and the fault injection device was switched off using the signal AL 7 (see Figure 11). The tested node was then restarted by the comparator node using AL 6 and a status message was sent via the serial ports (also collected by the workstation).

Upon restart, the workstation downloaded the application to the tested node. When the application had been restarted, the fault injection device was switched on using AL 7 and a new experimental run began.

## 4.4 Measurements

This section gives the method for experimental assessment of the fail-silence property of a MARS node when subjected to fault injection, as well as a precise definition of the predicates considered to perform the analysis. Several fault injection campaigns utilizing the three physical fault injection techniques described in Chapter 3 were conducted (see Section 5.1).

### 4.4.1 Experimental Assessment

Each campaign consisted of a number of experimental runs. During each experimental run a fault was injected into the tested node as described in Section 4.3; i.e. when an error occurred, the tested node was shut down by the comparator node to clear the error conditions for a new experimental run, and the power was then reinstalled and the tested node reloaded for the next experimental run.

An assessment of the fail-silence property was made by collecting the error detection information provided by the tested node and comparator node. This data collection was made by a program called ‘observe’ (running on the UNIX workstation), which continuously monitored the application and communication unit serial ports as well as the comparator node’s application unit serial port. The data delivered on these serial ports was saved by ‘observe’ in a single file on the workstation for each experimental campaign. Analysis of the information according to the predicates given in Section 4.4.2. was then made some time after the campaign had been conducted, using a two step process. First, extraction and restructuring of useful data from the collected data was carried out by a program called ‘extract’, after which the extracted data was used for statistical analysis by a program called ‘statcalc’. Several combinations of enabled/disabled EDMs were analysed in different campaigns (see Section 5.1) in order to study their impact on the fail-silence property.

Although these measurements were used for assessing the fail-silence coverage and the efficiency of the various EDMs of a MARS node, estimating the *real* coverage is exceedingly more complex. This is because the real fault set is usually not known in detail, and



even less is known about the probability of the occurrence of individual faults, although an estimate of the real coverage can be calculated as a weighted mean of the coverage factors obtained by different fault injection methods [Powell *et al.* 1995]. These weight factors are often very difficult or even impossible to calculate, however, due to a lack of knowledge about the real faults.

The fault injection techniques used in this study are therefore regarded as ‘benchmark’ techniques only, used for evaluating the relative efficiency of the various EDMs of the MARS system. The particular workload used in this study constitutes only a subset of the complete activity set, and the techniques and locations used for fault injection span only a subset of the complete fault set. The only way of estimating the real coverage is to use the MARS system in its intended environment, which is impractical if one wishes to estimate the value in a reasonable amount of time. However, by combining several fault injection techniques, more of the complete fault set should be sampled and the overlap between the fault set created by reality and the techniques should grow. Thus, the possibility of obtaining a higher confidence in the coverage value improves.

#### 4.4.2 Predicates

The tested node can be affected by four types of failures:

- (1) The EDMs within the node detect the error. This will cause it to stop sending messages on the MARS bus.
- (2) The node fails to deliver messages for one or several application cycles, but no error is detected by the node’s EDMs.
- (3) The node sends a message that is syntactically correct but has an erroneous content, a fail-silence violation in the value domain.
- (4) The node sends a message (erroneous or not) at an illegal point in time, a fail-silence violation in the time domain.

From these failure types, the following predicates (events) can be derived:

**Warmstart (WS):** A warmstart (reset) of the tested node occurs when one of the node’s EDMs detects an error (Internal WS), or when an incoming or outgoing link failure is detected by the top-layer mechanism for achieving fault tolerance (see Section 2.3) (External WS).

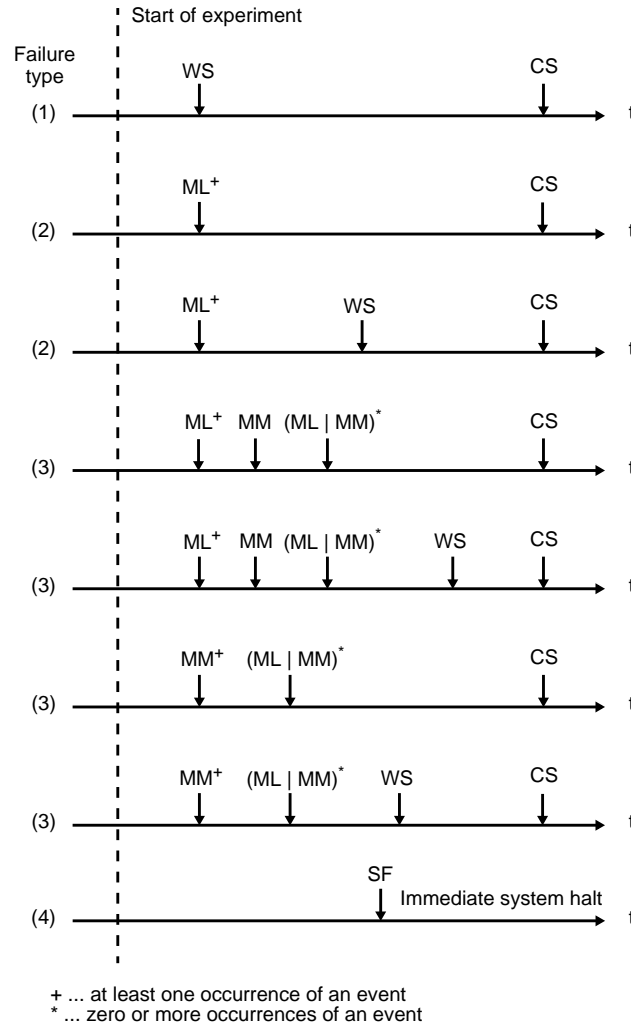
**Message loss (ML):** Occurs when a message is lost from the tested node, i.e. is not received by the comparator node.

**Message mismatch (MM):** Occurs when the contents of the messages delivered by the tested node and golden node differ at the comparator node.

**System failure (SF):** A system failure occurs when either the golden, data generation or comparator node fails.

**Coldstart (CS):** A coldstart (power on) of the tested node is made after each experiment

run, except when a system failure has occurred.



**Figure 12: Possible failure scenarios**

The first four predicates corresponds roughly to the failure types mentioned above (see also Figure 12). The assertion (occurrence) of the WS predicate in the data collected from the experiments indicates the usual case (failure type 1) in which the error is detected by the tested node itself. The ML predicate corresponds to failure type 2 and is not a fail-silence violation because no erroneous messages are sent, but the error is not detected by the EDMs in the tested node. The assertion of MM always corresponds to a fail-silence violation (failure type 3), regardless of other events. The SF predicate may be asserted when either a fail-silence violation in the time domain occurs (failure type 4) or when a hardware failure is experienced at a node other than the tested node. The CS predicate always indicates the end of each experimental run (i.e., the end of each data set).

Given the above failure types, the number of fail-silence violations is calculated as:

$$\#FS\ Viol. = \#Exp. \supseteq MM + \#Exp. \supseteq SF$$

where  $\#Exp. \supseteq X$  counts the number of experiments in which an  $X$ -type failure was diagnosed (i.e. predicate  $X$  was asserted).

## 5 Experimental Results

This chapter presents the results obtained from the fault injection campaigns conducted at each site using the experimental set-up described in Chapter 4. By studying the distribution of the error detections among the EDMs built into the MARS system, it is possible to indirectly analyse the error sets generated by the three techniques in order to determine whether or not the techniques are complementary, i.e. whether they generate different error sets. If they are complementary, it would mean that all three techniques could be applied in the validation of a fault-tolerant system in order to achieve higher confidence in the results.

To achieve as much similarity as possible among the error sets, so that the fault injection techniques could be objectively compared, faults were only injected inside, on the pins or in the vicinity of either the application unit CPU or the communication unit CPU of the tested node.

### 5.1 Experimental Combinations Used

Several fault injection campaigns using different combinations of EDMs activated in the tested node were used in the validation of the MARS system and for comparing the fault injection techniques (see Table 3). The following acronyms are used for the combinations: NOAM (no application level mechanisms, i.e. single execution and no message checksums used), SEMC (single execution and message checksums), DEMC (double execution and message checksums used, i.e. all EDMs activated), and TEMC (triple execution, message checksums; where an additional, third, time-redundant test execution of the control task was used for investigating the heavy-ion experiments, see Section 5.3). When the acronym ends with an ‘N’ (‘No NMIs’), the NMI EDMs of the tested node were deactivated as well.

Combination no.	Execution	Message Checksum	NMIs	Acronym
1	Single	No	No	NOAMN
2	Single	Yes	No	SEMCN
3	Double	Yes	No	DEMCN
4	Single	No	Yes	NOAM
5	Single	Yes	Yes	SEMC
6	Double	Yes	Yes	DEMC
7	Triple	Yes	Yes	TEMC

**Table 3: Experimental combinations used**

The fault injection was focused on either the application unit CPU or the communication unit CPU of the tested node. When the acronyms are used hereafter, they may therefore be suffixed by a letter. An ‘a’ after the acronym indicates that fault injection was focused

on the application unit CPU, and a ‘c’ that the communication unit CPU was the principal target.

## 5.2 Format of Presented Results

The results presented in this chapter are the distributions of the errors among the mechanisms that detected the error, divided into the following categories:

- *CPU* indicates errors detected by the CPU EDMs.
- *UEE* indicates errors causing unexpected exceptions to be raised, i.e. neither the CPU EDMs nor the EDMs provided by extra hardware were mapped to these exceptions.
- *NMI* indicates errors detected by the extra hardware in the unit (generating NMIs).
- *OS* indicates errors detected by the EDMs built into the operating system.
- *CGRTA* indicates errors detected by the Compiler Generated Run-Time Assertions.
- *Double exec.* indicates errors detected by comparing the messages produced by the two time-redundant executions of each task.
- *Checksum* indicates errors detected by the end-to-end checksum EDMs.
- *Other unit* indicates errors that caused error information to be given by the other (fault free) unit of the tested node only.
- *No error info.* indicates errors that did not produce any error information at all.
- *Triple execution* indicates errors detected by a comparison of the message produced by a third time redundant execution of the control task—which was provided with fixed input data used for verifying the double execution EDM—with the messages produced in the normal time redundant executions of this task. (Used only for the heavy-ion radiation technique).
- *Fail-silence violations* indicate errors that led to fail-silence violations.
- *Total no. of errors* indicates the number of errors observed in the combination considered.

The observed relative frequencies of the errors belonging to each category are given in the tables presenting the results for each fault injection technique in sequence using the combinations described in Section 5.1. In order to verify the results statistically, the 95% confidence intervals of the corresponding probabilities for the relative frequencies are given as well. If the observed number of errors is large, the relative frequencies are approximately normally distributed. The confidence intervals  $e$  can then be calculated as

$$e = \pm z_{\alpha/2} \sqrt{\frac{f_n(1-f_n)}{n}}$$

where  $n$  is the total number of observed errors of the combination considered,  $f_n$  the observed relative frequency whose confidence level should be calculated and  $z_{\alpha/2}$  calculated from the  $N(0,1)$  distribution, e.g.  $z_{\alpha/2}=1.9600$  for an interval with a 95% level of confidence.

The errors of the “Other unit” category, i.e. errors detected by the other (fault free) unit of the tested node, have been divided into similar categories and are presented in a similar way in separate tables underneath the main ones.

### 5.3 Results of Heavy-Ion Radiation

The heavy-ion radiation technique was used for fault injection of either the application unit CPU or communication unit CPU. The fault injection was performed using the miniature vacuum chamber described in Section 3.1, containing the irradiated CPU and the Cf-252 source. As the irradiated ICs are CMOS circuits, they were protected from latch-ups by a current guard that turned off the power to the CPUs when the current exceeded a threshold value. All observed latch-ups have been censored out of the data given here (about 30-50% of all errors depending on which unit was fault injected, the pressure in the miniature vacuum chamber, the amount of time the same chip had been irradiated and the activity of the Cf-252 source). Figure 13 shows how the latch-up intensity varied during the experiments (the theoretical values of the Cf-252 source activity, which was measured as 43 kBq as of April 4th 1991, are also shown). The general trend seems to be that the latch-up intensity *increased* over time during the first campaigns, contrary to what is expected since the activity of the Cf-252 source *decreases*. Whether this has to do with the chip becoming increasingly more contaminated or that the energy spectrum of the Cf-252 source varies (or if another cause for this behaviour exists) is not yet clear. According to [Johansson1993], the Cf-252 source used in the experiments was either badly manufactured or had accidentally been touched, since measurements of the energy spectrum of the source showed it to be very different from what was expected.

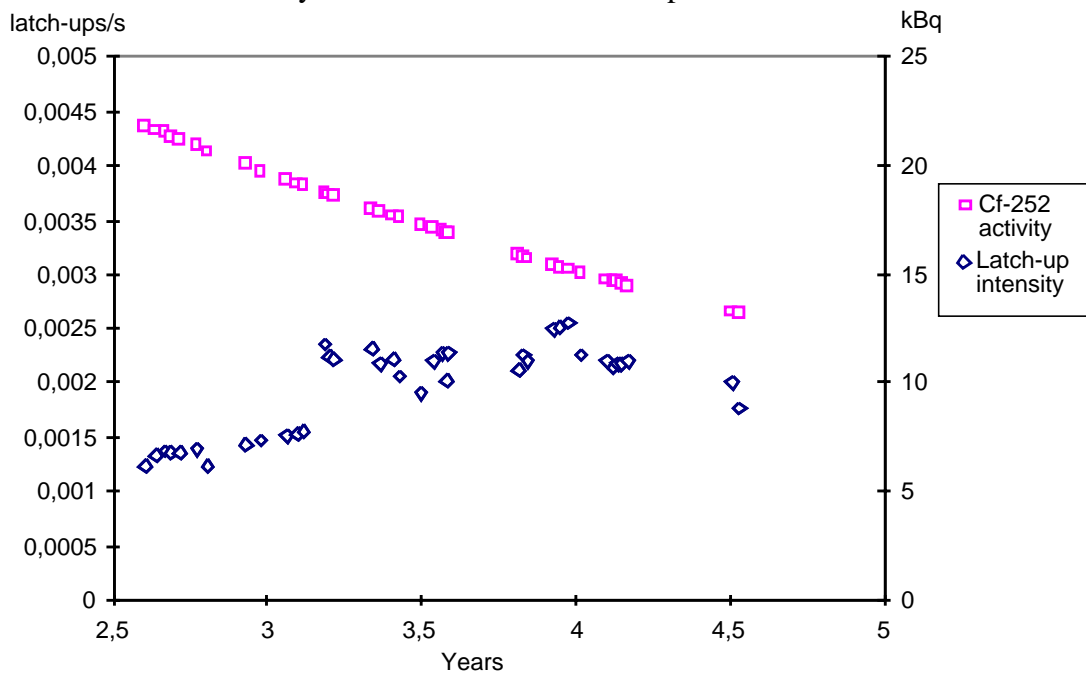


Figure 13: Latch-up intensity vs. Cf-252 activity

Error Detection Mechanisms		application unit CPU irradiated						comm. unit CPU irradiated			
		NOAMNa		SEMCNa		DEMCNa		NOAMNc		DEMCNc	
		% (95% conf.)		% (95% conf.)		% (95% conf.)		% (95% conf.)		% (95% conf.)	
Level 1 Hardware	CPU	54.9%	±1.9%	58.2%	±2.0%	54.3%	±2.0%	55.2%	±1.6%	56.0%	±2.0%
	UEE	11.2%	±1.2%	11.1%	±1.3%	10.3%	±1.2%	9.9%	±1.0%	10.7%	±1.2%
	<i>Subtotal</i>	<i>66.1%</i>	<i>±1.8%</i>	<i>69.4%</i>	<i>±1.9%</i>	<i>64.6%</i>	<i>±1.9%</i>	<i>65.1%</i>	<i>±1.6%</i>	<i>66.7%</i>	<i>±1.9%</i>
Level 2 System software	OS	7.2%	±1.0%	8.2%	±1.1%	7.8%	±1.1%	15.2%	±1.2%	6.3%	±1.0%
	CGRTA	0.3%	±0.2%	0.3%	±0.2%	0.2%	±0.2%	0.2%	±0.1%	0.5%	±0.3%
	<i>Subtotal</i>	<i>7.5%</i>	<i>±1.0%</i>	<i>8.5%</i>	<i>±1.1%</i>	<i>8.0%</i>	<i>±1.1%</i>	<i>15.4%</i>	<i>±1.2%</i>	<i>6.8%</i>	<i>±1.0%</i>
Level 3 Application level	Double exec.	—	—	—	—	1.0%	±0.4%	—	—	0.8%	±0.4%
	Checksum	—	—	—	—	±0.5%	2.8%	±0.7%	—	—	2.8%
	<i>Subtotal</i>	—	—	<i>1.5%</i>	<i>±0.5%</i>	<i>3.8%</i>	<i>±0.8%</i>	—	—	<i>3.7%</i>	<i>±0.8%</i>
Other	Other unit	12.0%	±1.3%	12.1%	±1.3%	11.3%	±1.3%	7.8%	±0.9%	8.8%	±1.1%
	No error info.	13.1%	±1.3%	8.1%	±1.1%	12.2%	±1.3%	11.4%	±1.0%	12.6%	±1.3%
	<i>Subtotal</i>	<i>25.1%</i>	<i>±1.7%</i>	<i>20.2%</i>	<i>±1.6%</i>	<i>23.5%</i>	<i>±1.7%</i>	<i>19.3%</i>	<i>±1.3%</i>	<i>21.4%</i>	<i>±1.6%</i>
Fail-silence violations		1.3%	±0.4%	0.4%	±0.3%	0.1%	±0.1%	0.3%	±0.2%	0%	—
<i>Total no. of errors</i>		<i>2565</i>		<i>2342</i>		<i>2347</i>		<i>3555</i>		<i>2458</i>	

(a) Detection by the EDMs of the unit to which the faulted ICs belong

Error Detection Mechanisms		application unit CPU irradiated						comm. unit CPU irradiated			
		NOAMNa		SEMCNa		DEMCNa		NOAMNc		DEMCNc	
		% (95% conf.)		% (95% conf.)		% (95% conf.)		% (95% conf.)		% (95% conf.)	
Level 1 Hardware	CPU	0%	—	0%	—	0%	—	0%	—	0%	—
	UEE	0.2%	±0.2%	0.1%	±0.1%	0.2%	±0.2%	<0.1%	±0.1%	<0.1%	±0.1%
	<i>Subtotal</i>	<i>0.2%</i>	<i>±0.2%</i>	<i>0.1%</i>	<i>±0.1%</i>	<i>0.2%</i>	<i>±0.2%</i>	<i>&lt;0.1%</i>	<i>±0.1%</i>	<i>&lt;0.1%</i>	<i>±0.1%</i>
Level 2 System software	OS	11.8%	±1.2%	12.0%	±1.3%	11.1%	±1.3%	7.8%	±0.9%	8.6%	±1.1%
	CGRTA	0%	—	0%	—	0%	—	<0.1%	±0.1%	0%	—
	<i>Subtotal</i>	<i>11.8%</i>	<i>±1.2%</i>	<i>12.0%</i>	<i>±1.3%</i>	<i>11.1%</i>	<i>±1.3%</i>	<i>7.8%</i>	<i>±0.9%</i>	<i>8.6%</i>	<i>±1.1%</i>
Level 3 Application level	Double exec.	—	—	—	—	0%	—	—	—	0%	—
	Checksum	—	—	0%	—	0%	—	—	—	0.2%	±0.2%
	<i>Subtotal</i>	—	—	<i>0%</i>	—	<i>0%</i>	—	—	—	<i>0.2%</i>	<i>±0.2%</i>

(b) Detection by the EDMs of the other unit (detail of “Other unit” entry in Table (a) above)

**Table 4: Results of heavy-ion radiation without activated NMI EDMs**

Table 4 shows the distribution of error detections among the various EDMs for each of the irradiated CPUs and those combinations given in Table 3 for which no NMI EDMs were activated in the tested node (combinations 1 to 3). Table 5 shows the results obtained when the NMI EDMs were activated (combinations 4 to 7). The observed relative frequencies are given together with the corresponding confidence intervals.

The hardware EDMs, in particular the CPU mechanisms, detected most of the errors, which is not surprising since the faults were injected into the CPU. The proportion of errors detected by the hardware EDMs is larger for faults injected into the communication unit CPU than for faults injected into the application unit CPU, except when the NMI EDMs were deactivated, when the proportions are similar. In particular, the coverage of the NMI EDMs is highest when the application unit CPU was irradiated. Unexpected exceptions (UEE) occurred with a frequency of about 15% in the combinations in which all NMI EDMs were used vs. 10% when no NMI EDMs were used. This is somewhat surprising since one would expect the opposite, i.e. a higher frequency of UEEs to occur when the NMIs were not used than when they were used. Unfortunately, the lack of observability in the experiments makes it difficult to speculate on why this behaviour was observed.

Errors detected by the OS EDMs dominate for the software EDMs. For application level EDMs, the message checksum EDMs dominate.

## 5 Experimental Results

Error Detection Mechanisms		application unit CPU irradiated								communication unit CPU irradiated					
		NOAMa		SEMCa		DEMCa		TEMCa		NOAMc		SEMCc		DEMCc	
		% (95% conf.)		% (95% conf.)		% (95% conf.)		% (95% conf.)		% (95% conf.)		% (95% conf.)		% (95% conf.)	
Level 1 Hardware	CPU	47.7%	±1.1%	49.0%	±1.8%	47.4%	±1.0%	51.3%	±1.4%	44.9%	±2.0%	43.2%	±1.8%	43.3%	±2.0%
	UEE	15.0%	±0.8%	16.0%	±1.3%	15.2%	±0.7%	14.7%	±1.0%	14.6%	±1.4%	14.1%	±1.3%	13.4%	±1.4%
	NMI	7.0%	±0.6%	6.0%	±0.9%	6.3%	±0.5%	5.7%	±0.6%	20.2%	±1.6%	19.6%	±1.4%	19.9%	±1.6%
	<i>Subtotal</i>	69.7%	±1.0%	71.0%	±1.7%	68.9%	±1.0%	71.7%	±1.2%	79.6%	±1.6%	76.9%	±1.5%	76.6%	±1.7%
Level 2 System software	OS	7.8%	±0.6%	7.7%	±1.0%	7.6%	±0.5%	5.4%	±0.6%	3.6%	±0.7%	4.9%	±0.8%	5.3%	±0.9%
	CGRTA	1.0%	±0.2%	0.1%	±0.1%	0.3%	±0.1%	0.7%	±0.2%	0.4%	±0.2%	0.2%	±0.2%	0.5%	±0.3%
	<i>Subtotal</i>	8.8%	±0.6%	7.8%	±1.0%	7.9%	±0.6%	6.2%	±0.7%	4.0%	±0.8%	5.1%	±0.8%	5.8%	±0.9%
Level 3 Application level	Double exec.	—	—	—	—	0.8%	±0.2%	1.1%	±0.3%	—	—	—	—	0.5%	±0.3%
	Checksum	—	—	2.4%	±0.6%	2.7%	±0.3%	4.6%	±0.6%	—	—	1.6%	±0.5%	3.1%	±0.7%
	<i>Subtotal</i>	—	—	2.4%	±0.6%	3.6%	±0.4%	5.7%	±0.6%	—	—	1.6%	±0.5%	3.5%	±0.7%
Other	Other unit	14.0%	±0.8%	13.2%	±1.2%	14.3%	±0.7%	11.3%	±0.9%	13.8%	±1.4%	13.8%	±1.2%	12.0%	±1.3%
	No error info.	5.1%	±0.5%	4.2%	±0.7%	4.8%	±0.4%	4.3%	±0.6%	2.5%	±0.6%	2.5%	±0.6%	2.1%	±0.6%
	<i>Subtotal</i>	19.1%	±0.9%	17.5%	±1.4%	19.1%	±0.8%	15.6%	±1.0%	16.3%	±1.5%	16.3%	±1.3%	14.1%	±1.4%
Triple execution		—	—	—	—	—	—	0.8%	±0.3%	—	—	—	—	—	—
Fail-silence violations		2.4%	±0.3%	1.3%	±0.4%	0.5%	±0.1%	0%	—	<0.1%	±0.1%	0%	—	0%	—
<i>Total no. of errors</i>		7825		2877		9036		5016		2479		2943		2437	

**(a) Detection by the EDMs of the unit to which the faulted ICs belong**

Error Detection Mechanisms		application unit CPU irradiated								communication unit CPU irradiated					
		NOAMa		SEMCa		DEMCa		TEMCa		NOAMc		SEMCc		DEMCc	
		% (95% conf.)		% (95% conf.)		% (95% conf.)		% (95% conf.)		% (95% conf.)		% (95% conf.)		% (95% conf.)	
Level 1 Hardware	CPU	0%	—	0%	—	0%	—	0%	—	0%	—	0%	—	0%	—
	UEE	<0.1%	±0.0%	0.1%	±0.1%	0%	—	0.1%	±0.1%	0%	—	0%	—	<0.1%	±0.1%
	NMI	2.5%	±0.3%	2.0%	±0.5%	2.7%	±0.3%	2.1%	±0.4%	4.7%	±0.8%	5.0%	±0.8%	4.2%	±0.8%
	<i>Subtotal</i>	2.6%	±0.4%	2.1%	±0.5%	2.7%	±0.3%	2.1%	±0.4%	4.7%	±0.8%	5.0%	±0.8%	4.3%	±0.8%
Level 2 System software	OS	11.4%	±0.7%	11.2%	±1.2%	11.6%	±0.7%	9.2%	±0.8%	8.9%	±1.1%	8.8%	±1.0%	7.8%	±1.1%
	CGRTA	0%	—	0%	—	0%	—	0%	—	0%	—	0%	—	0%	—
	<i>Subtotal</i>	11.4%	±0.7%	11.2%	±1.2%	11.6%	±0.7%	9.2%	±0.8%	8.9%	±1.1%	8.8%	±1.0%	7.8%	±1.1%
Level 3 Application level	Double exec.	—	—	—	—	0%	—	0%	—	—	—	—	—	0%	—
	Checksum	—	—	0%	—	0%	—	0%	—	—	—	0%	—	0%	—
	<i>Subtotal</i>	—	—	0%	—	0%	—	0%	—	—	—	0%	—	0%	—

**(b) Detection by the EDMs of the other unit (detail of “Other unit” entry in Table (a) above)**

**Table 5: Results of heavy-ion radiation with activated NMI EDMs**

The amount of errors that did not produce any error information is in the range of 10% for the combinations without NMIs vs. 5% when the NMIs were used, which is not surprising since faults that are not caught by NMI EDMs may cause an unexpected behaviour of the rest of the system (i.e. the system is designed to be used with the NMIs).

The percentage of fail-silence violations was between 2.4% and 0.5% for the NOAMa, SEMCa and DEMCa combinations, and between 1.3% and 0.1% for the NOAMNa, SEMCNa and DEMCNa combinations. This is clearly an anomaly since one would expect more fail-silence violations when less EDMs are activated. The anomaly is also confirmed statistically, as it can not even be explained by taking into account the confidence interval figures. However, the number of fail-silence violations is always lower for SEMC than for NOAM, and even lower for DEMC, as would be expected. All fail-silence violations were in the value domain except for combination NOAMNc, for which three system failures were observed due to fail-silence violations in the time domain. This could be observed by the assertion of the SF predicate (see Section 4.4.2) in the data collected from the experiments (assuming there were no failures of other nodes than the tested node).

Fewer fail-silence violations were always observed when faults were injected into the communication unit CPU than when the application unit CPU was fault-injected. This is true regardless of whether or not the NMIs were activated.



The observation of fail-silence violations for the combination for which all EDMs were activated (DEMCa) was unexpected since all transient errors should be detected by the double execution feature of the MARS system. Two different hypotheses may explain this behaviour. The first is obvious, a design error in the MARS system. However, as no other fault injection technique caused any fail-silence violations for this combination (see Sections 5.4 and 5.5), another heavy-ion radiation-specific hypothesis may explain the behaviour. This second hypothesis is that an undetected latch-up caused the same incorrect result to be produced by both executions of the control task.

To further investigate this hypothesis, the TEMCa combination was applied, using a third time redundant execution of the control task which was provided with fixed input data for which the results were known. This made it possible to detect errors by comparing the produced results with the correct results. This mechanism, which can be viewed as an on-line test program, where the third execution is included for comparison purposes only (i.e. its results are never used by the application), would detect any semi-permanent fault, such as the one suggested by the latch-up hypothesis. If fail-silence violations would still have been observed, the latch-up hypothesis could be rejected and a design error in the MARS system would be the next logical explanation.

The results show that no fail-silence violations occurred for the TEMCa combination. As Table 5-a shows, 0.8% of the errors (42 errors) were detected by the third execution of the control task. This supports the latch-up hypothesis. However, the experimental set-up used in the experiments does not provide sufficient observability to fully prove the hypothesis. The absence of fail-silence violations may merely be an effect of the change of software configuration caused by the switch from DEMCa to TEMCa, and the errors detected by the third execution may have been caused by regular transients (which further analysis of the collected data definitely shows to have occurred for six errors). Verification of the latch-up hypothesis would require the use of a logic analyser in order to study the program flow and behaviour of the microprocessor in detail. The development of such a set-up using the golden chip technique for synchronizing logic analyser data collection with the occurrence of an error and for measuring error detection latency, had to be discontinued. This was due to severe adaption problems when moving the target CPU to the separate comparator card that was needed to hold the target and golden (reference) CPUs and to which the logic analyser was connected.

The OS and NMI EDMs dominate the detections made by the other unit of the tested node. The communication between the two units is carried out entirely via FIFO buffers, and nearly all of these detections were made by EDMs signalling empty FIFO. (An empty FIFO can be detected both by the operating system and a special NMI mechanism.)

Experiments were also conducted using a different sample of the target CPU in order to validate the statistical reproducibility of the heavy-ion radiation technique. The results of these campaigns are shown in Table 6. These results are very similar to those obtained in the original campaigns which indeed validates the statistical reproducibility of the heavy-ion radiation technique. Only one major discrepancy exists, that of the number of fail-silence violations observed. Fewer than one half the amount of fail-silence violations were

observed in these later experiments. This is consistent with what was observed in the original campaigns, however, since the amount of fail-silence violations was lower for campaigns made later (i.e. the campaigns with the NMIs switched off). See Section 6.3 for a more thorough discussion on the number of fail-silence violations observed.

Error Detection Mechanisms		application unit CPU irradiated			
		NOAMNa % (95% conf.)	DEMCNa % (95% conf.)	SEMCA % (95% conf.)	DEMCa % (95% conf.)
Level 1 Hardware	CPU	61.6% ±3.0%	58.3% ±6.7%	54.2% ±3.1%	53.1% ±1.3%
	UEE	10.5% ±1.9%	9.2% ±4.0%	15.9% ±2.3%	15.9% ±1.0%
	NMI	—	—	6.6% ±1.6%	5.7% ±0.6%
	<i>Subtotal</i>	<i>72.1% ±2.8%</i>	<i>67.5% ±6.4%</i>	<i>76.7% ±2.7%</i>	<i>74.7% ±1.2%</i>
Level 2 System software	OS	5.5% ±1.4%	10.2% ±4.1%	5.2% ±1.4%	6.1% ±0.6%
	CGRTA	0.3% ±0.3%	0%	0%	0.3% ±0.1%
	<i>Subtotal</i>	<i>5.8% ±1.5%</i>	<i>10.2% ±4.1%</i>	<i>5.2% ±1.4%</i>	<i>6.4% ±0.7%</i>
Level 3 Application level	Double exec.	—	0.5% ±0.9%	—	0.8% ±0.2%
	Checksum	—	1.4% ±1.6%	1.1% ±0.7%	1.9% ±0.4%
	<i>Subtotal</i>	—	<i>1.9% ±1.9%</i>	<i>1.1% ±0.7%</i>	<i>2.7% ±0.4%</i>
Other	Other unit	13.8% ±2.1%	13.6% ±4.7%	13.0% ±2.1%	11.8% ±0.9%
	No error info.	7.7% ±1.7%	6.8% ±3.4%	3.5% ±1.1%	4.3% ±0.5%
	<i>Subtotal</i>	<i>21.5% ±2.6%</i>	<i>20.4% ±5.5%</i>	<i>16.5% ±2.3%</i>	<i>16.1% ±1.0%</i>
Fail-silence violations		0.6% ±0.5%	0%	0.5% ±0.4%	<0.1% ±0.1%
<i>Total number of errors</i>		<i>988</i>	<i>206</i>	<i>977</i>	<i>5470</i>

(a) Detection by the EDMs of the unit to which the faulted ICs belong

Error Detection Mechanisms		application unit CPU irradiated			
		NOAMNa % (95% conf.)	DEMCNa % (95% conf.)	SEMCA % (95% conf.)	DEMCa % (95% conf.)
Level 1 Hardware	CPU	0%	0%	0%	0%
	UEE	0.1% ±0.2%	0.5% ±0.9%	0%	0.1% ±0.1%
	NMI	—	—	2.6% ±1.0%	2.3% ±0.4%
	<i>Subtotal</i>	<i>0.1% ±0.2%</i>	<i>0.5% ±0.9%</i>	<i>2.6% ±1.0%</i>	<i>2.4% ±0.4%</i>
Level 2 System software	OS	13.7% ±2.1%	13.1% ±4.6%	10.4% ±1.9%	9.3% ±0.5%
	CGRTA	0%	0%	0%	0%
	<i>Subtotal</i>	<i>13.7% ±2.1%</i>	<i>13.1% ±4.6%</i>	<i>10.4% ±1.9%</i>	<i>9.3% ±0.5%</i>
Level 3 Application level	Double exec.	—	0%	—	0%
	Checksum	—	0%	0%	0%
	<i>Subtotal</i>	—	<i>0%</i>	<i>0%</i>	<i>0%</i>

(b) Detection by the EDMs of the other unit (detail of "Other unit" entry in Table (a) above)

Table 6: Results of heavy-ion radiation using a different sample of the target CPU

## 5.4 Results of Pin-Level Injection

In the pin-level fault injection campaigns, the MESSALINE tool was applied with the following configuration:

- forcing was used,
- one IC at a time was fault injected with a multiplicity, i.e. maximum number of pins faulted at the same time, of  $mx=3$ ,
- the  $mx$  faulted pins selected were uniformly chosen from all possible combinations of  $mx$  pins,
- stuck-at 0 and 1 fault models were used (all 0-1 combinations of  $mx$  pins were considered equally probable),
- both transient and intermittent (series of transients) faults were injected to simplify the

- comparison with the heavy-ion radiation and EMI techniques, and
- the duration of each fault was randomly chosen in the [1  $\mu$ s, 10  $\mu$ s] range.

Since the forcing technique was used, the pins of all ICs connected to the same signal line (equipotential line) as a fault injected pin were affected by the fault as well (see Section 3.2). To simplify access to the pins of either the application unit CPU or the communication unit CPU of the tested node, buffer ICs connected to the CPUs were selected as target ICs.

Table 7 shows the results of pin-level fault injection when all NMI EDMs were deactivated, and Table 8 shows the results obtained when all NMIs were activated.

Error Detection Mechanisms		ICs belonging to the application unit					
		NOAMNa		SEMCNa		DEMCNa	
		% (95% conf.)		% (95% conf.)		% (95% conf.)	
Level 1 Hardware	CPU	28.3%	$\pm 3.3\%$	41.2%	$\pm 4.0\%$	43.3%	$\pm 6.4\%$
	UEE	4.2%	$\pm 1.5\%$	4.9%	$\pm 1.8\%$	6.5%	$\pm 3.2\%$
	<i>Subtotal</i>	<i>32.5%</i>	<i><math>\pm 3.4\%</math></i>	<i>46.1%</i>	<i><math>\pm 4.1\%</math></i>	<i>49.8%</i>	<i><math>\pm 6.4\%</math></i>
Level 2 System software	OS	51.2%	$\pm 3.7\%$	37.7%	$\pm 4.0\%$	35.9%	$\pm 6.2\%$
	CGRTA	1.8%	$\pm 1.0\%$	0%	—	0%	—
	<i>Subtotal</i>	<i>53.0%</i>	<i><math>\pm 3.7\%</math></i>	<i>37.7%</i>	<i><math>\pm 4.0\%</math></i>	<i>35.9%</i>	<i><math>\pm 6.2\%</math></i>
Level 3 Application level	Double exec.	—	—	—	—	0%	—
	Checksum	—	—	2.5%	$\pm 1.3\%$	3.5%	$\pm 2.4\%$
	<i>Subtotal</i>	<i>—</i>	<i>—</i>	<i>2.5%</i>	<i><math>\pm 1.3\%</math></i>	<i>3.5%</i>	<i><math>\pm 2.4\%</math></i>
Other	Other unit	3.3%	$\pm 1.3\%$	2.6%	$\pm 1.3\%$	0%	—
	No error info.	10.6%	$\pm 2.3\%$	10.7%	$\pm 2.5\%$	10.8%	$\pm 4.0\%$
	<i>Subtotal</i>	<i>13.9%</i>	<i><math>\pm 2.5\%</math></i>	<i>13.4%</i>	<i><math>\pm 2.8\%</math></i>	<i>10.8%</i>	<i><math>\pm 4.0\%</math></i>
Fail-silence violations		0.6%	$\pm 0.5\%$	0.4%	$\pm 0.5\%$	0%	—
<i>Total no. of errors</i>		717		568		231	

(a) Detection by the EDMs of the unit to which the faulted ICs belong

Error Detection Mechanisms		ICs belonging to the application unit					
		NOAMNa		SEMCNa		DEMCNa	
		% (95% conf.)		% (95% conf.)		% (95% conf.)	
Level 1 Hardware	CPU	0.7%	$\pm 0.6\%$	0.2%	$\pm 0.3\%$	0%	—
	UEE	0.7%	$\pm 0.6\%$	1.1%	$\pm 0.8\%$	0%	—
	<i>Subtotal</i>	<i>1.4%</i>	<i><math>\pm 0.9\%</math></i>	<i>1.2%</i>	<i><math>\pm 0.9\%</math></i>	<i>0%</i>	<i>—</i>
Level 2 System software	OS	1.8%	$\pm 1.0\%$	1.4%	$\pm 1.0\%$	0%	—
	CGRTA	0.1%	$\pm 0.3\%$	0%	—	0%	—
	<i>Subtotal</i>	<i>2.0%</i>	<i><math>\pm 1.0\%</math></i>	<i>1.4%</i>	<i><math>\pm 1.0\%</math></i>	<i>0%</i>	<i>—</i>
Level 3 Application level	Double exec.	—	—	—	—	0%	—
	Checksum	—	—	0%	—	0%	—
	<i>Subtotal</i>	<i>—</i>	<i>—</i>	<i>0%</i>	<i>—</i>	<i>0%</i>	<i>—</i>

(b) Detection by the EDMs of the other unit (detail of "Other unit" entry in Table (a) above)

**Table 7: Results of pin-level injection without activated NMI EDMs**

The hardware EDMs dominate the detections made when using pin-forcing. The percentage of errors detected by NMI mechanisms is exceptionally large (about 75%) when the NMI EDMs were activated, while most detections were made by the CPU mechanisms or OS mechanisms (about 40% each) when the NMIs were deactivated (except for the NOAMNa combination for which the OS mechanisms dominate entirely). More UEEs were triggered when faults were injected into the communication unit than when the application unit was fault injected. For the CPU mechanisms, the reverse was observed, i.e., fewer CPU EDMs were triggered when faults were injected into the communication unit than when the application unit was fault-injected.

For software EDMs, the detections made by the operating system EDMs dominate (from 4% when NMIs were used up to 50% when the NMIs were switched off).

For the application level EDMs, only detections made by message checksum EDMs were made, as no double execution EDMs were triggered at all.

The percentage of fail-silence violations was observed to be as large as 0.6% when no NMIs were used and was never greater than 0.2% when the NMIs were activated. No fail-silence violations were observed when all the EDMs were activated (DEMCa and DEMCc combinations). In fact, no fail-silence violations were observed in any case when all the application level EDMs were activated, i.e. even for the DEMCNa combination, although the statistical material for this combination is quite small (only 231 errors collected).

When NMIs were used, the supplementary detections made by the fault free unit of the tested node are dominated by NMI error detections, while the OS error detections dominate when the NMIs were deactivated. A greater number of CPU detections and UEEs occurred in the latter case as well.

Error Detection Mechanisms		ICs belonging to the application unit						ICs belonging to the communication unit					
		NOAMa		SEMCA		DEMCa		NOAMc		SEMCC		DEMCc	
		% (95% conf.)		% (95% conf.)		% (95% conf.)		% (95% conf.)		% (95% conf.)		% (95% conf.)	
Level 1 Hardware	CPU	6.5%	±1.5%	4.6%	±1.3%	4.1%	±1.2%	3.5%	±1.1%	3.6%	±1.1%	1.7%	±0.7%
	UEE	5.8%	±1.4%	7.8%	±1.7%	6.2%	±1.5%	10.5%	±1.8%	9.0%	±1.8%	10.5%	±1.8%
	NMI	77.0%	±2.5%	75.6%	±2.7%	78.1%	±2.6%	74.0%	±2.6%	75.6%	±2.6%	73.8%	±2.5%
	<i>Subtotal</i>	<i>89.3%</i>	<i>±1.8%</i>	<i>88.0%</i>	<i>±2.0%</i>	<i>88.4%</i>	<i>±2.0%</i>	<i>87.9%</i>	<i>±1.9%</i>	<i>88.1%</i>	<i>±2.0%</i>	<i>86.1%</i>	<i>±2.0%</i>
Level 2 System software	OS	4.1%	±1.2%	4.7%	±1.3%	4.6%	±1.3%	3.9%	±1.1%	4.0%	±1.2%	4.7%	±1.2%
	CGRTA	0.5%	±0.4%	0.1%	±0.2%	0%	—	0%	—	0%	—	0%	—
	<i>Subtotal</i>	<i>4.6%</i>	<i>±1.2%</i>	<i>4.8%</i>	<i>±1.4%</i>	<i>4.6%</i>	<i>±1.3%</i>	<i>3.9%</i>	<i>±1.1%</i>	<i>4.0%</i>	<i>±0.4%</i>	<i>4.7%</i>	<i>±1.2%</i>
Level 3 Application level	Double exec.	—	—	—	—	0%	—	—	—	—	—	0%	—
	Checksum	—	—	0.6%	±0.5%	0.6%	±0.5%	—	—	0.4%	±0.4%	0.4%	±0.4%
	<i>Subtotal</i>	—	—	<i>0.6%</i>	<i>±0.5%</i>	<i>0.6%</i>	<i>±0.5%</i>	—	—	<i>0.4%</i>	<i>±0.4%</i>	<i>0.4%</i>	<i>±0.4%</i>
Other	Other unit	0.4%	±0.4%	0.7%	±0.5%	0.4%	±0.4%	2.1%	±0.8%	1.6%	±0.8%	2.2%	±0.9%
	No error info.	5.6%	±1.4%	5.6%	±1.4%	6.0%	±1.5%	6.0%	±1.4%	5.7%	±1.4%	6.5%	±1.4%
	<i>Subtotal</i>	<i>6.0%</i>	<i>±1.4%</i>	<i>6.3%</i>	<i>±1.5%</i>	<i>6.4%</i>	<i>±1.5%</i>	<i>8.1%</i>	<i>±1.6%</i>	<i>7.3%</i>	<i>±1.6%</i>	<i>8.8%</i>	<i>±1.6%</i>
Fail-silence violations		0.1%	±0.2%	0.2%	±0.3%	0%	—	0.1%	±0.2%	0.2%	±0.3%	0%	—
<i>Total no. of errors</i>		<i>1102</i>		<i>970</i>		<i>999</i>		<i>1099</i>		<i>1012</i>		<i>1162</i>	

(a) Detection by the EDMs of the unit to which the faulted ICs belong

Error Detection Mechanisms		ICs belonging to the application unit						ICs belonging to the communication unit					
		NOAMa		SEMCA		DEMCa		NOAMc		SEMCC		DEMCc	
		% (95% conf.)		% (95% conf.)		% (95% conf.)		% (95% conf.)		% (95% conf.)		% (95% conf.)	
Level 1 Hardware	CPU	0%	—	0%	—	0%	—	0%	—	0%	—	0%	—
	UEE	0%	—	0%	—	0.2%	±0.3%	0%	—	0%	—	0.2%	±0.2%
	NMI	0.4%	±0.4%	0.5%	±0.5%	0.2%	±0.3%	2.1%	±0.8%	1.6%	±0.8%	2.1%	±0.8%
	<i>Subtotal</i>	<i>0.4%</i>	<i>±0.4%</i>	<i>0.5%</i>	<i>±0.5%</i>	<i>0.4%</i>	<i>±0.4%</i>	<i>2.1%</i>	<i>±0.8%</i>	<i>1.6%</i>	<i>±0.8%</i>	<i>2.2%</i>	<i>±0.9%</i>
Level 2 System software	OS	0%	—	0.2%	±0.3%	0%	—	0%	—	0%	—	0%	—
	CGRTA	0%	—	0%	—	0%	—	0%	—	0%	—	0%	—
	<i>Subtotal</i>	<i>0%</i>	—	<i>0.2%</i>	—	<i>0%</i>	—	<i>0%</i>	—	<i>0%</i>	—	<i>0%</i>	—
Level 3 Application level	Double exec.	—	—	—	—	0%	—	—	—	—	—	0%	—
	Checksum	—	—	0%	—	0%	—	—	—	0%	—	0%	—
	<i>Subtotal</i>	—	—	<i>0%</i>	—	<i>0%</i>	—	—	—	<i>0%</i>	—	<i>0%</i>	—

(b) Detection by the EDMs of the other unit (detail of "Other unit" entry in Table (a) above)

**Table 8: Results of pin-level injection with activated NMI EDMs**

## 5.5 Results of EMI

Several fault injection campaigns using the two alternatives for EMI fault injection described in Section 3.3 were carried out. Table 9 shows the results obtained in the most informative campaigns, when the alternatives using antenna wires were applied. When the probe technique was applied without antennas, the spread in the detections among the error detection mechanisms was very low, suggesting further refinement of the technique as it is clearly not useful for validating fault-tolerant computer systems. (Almost no other mechanisms other than spurious interrupts and unexpected exceptions were triggered using this alternative.) All the campaigns shown in Table 9 were conducted using activated NMIs.

The first campaign in Table 9 (NOAMa) used the two conducting plates with a burst frequency of 2.5 kHz, negative burst polarity, and a voltage of 230 V. The antenna wires were mounted on the *low*-EPROM (i.e. the EPROM containing the eight least significant bits of each 16 bit code or data word) of the application unit, thus disturbing the address bus and the eight least significant bits of the data bus. The second campaign (NOAMc) used the same EMI conditions as the first, but the antenna wires were instead connected to the *low*-EPROM of the communication unit. The SEMCa campaign used the probe with bursts characterized by a frequency of 10 kHz, negative polarity and a voltage of 300 V. Wires were connected to the *low*-EPROM of the application unit. The DEMCa(1) campaign used plates with antennas mounted on the *low*-EPROM of the application unit. The frequency of the EMI burst was 2.5 kHz, positive polarity was used and the voltage was 300 V. The DEMCa(2) campaign used the two plates with EMI bursts of 2.5 kHz frequency, negative polarity and a voltage of 230 V. The wires were connected to the *low*-EPROM of the application unit.

The first two campaigns in Table 9 were made with an early version of the experimental set-up, where no supplementary detections by the other fault free unit of the tested node were collected. This explains the large amount of errors without error information for these two campaigns.

The hardware EDMs dominate for all combinations except for the DEMCa(2) campaign which shows a radically different distribution dominated by OS EDMs. This demonstrates the difficulties in statistically reproducing results when this technique is used. Even when the same EMI conditions were used as in the NOAMa and NOAMc campaigns, the results are very dissimilar. Whether this has to do with the distance between the plates and the computer board, the orientation of the plates compared with the computer board or antenna wires, synchronization with the system activity, or any other parameters that must be taken into account remains to be addressed by further study. The DEMCa(2) campaign shows, however, that no fail-silence violations occurred when all EDMs were activated and as many as 6093 errors were observed.

Error Detection Mechanisms		fault-injection with antennas									
		NOAMa		NOAMc		SEMCa		DEMCa(1)		DEMCa(2)	
		% (95% conf.)		% (95% conf.)		% (95% conf.)		% (95% conf.)		% (95% conf.)	
Level 1 Hardware	CPU	76.4%	±2.1%	72.0%	±2.2%	76.6%	±5.2%	76.2%	±2.1%	2.2%	±0.4%
	UEE	0.7%	±0.4%	0.7%	±0.4%	3.2%	±2.2%	1.1%	±0.5%	0.2%	±0.1%
	NMI	3.1%	±0.9%	2.9%	±0.8%	7.1%	±3.2%	4.2%	±1.0%	11.4%	±0.8%
	<i>Subtotal</i>	<i>80.2%</i>	<i>±2.0%</i>	<i>75.6%</i>	<i>±2.1%</i>	<i>86.9%</i>	<i>±4.2%</i>	<i>81.5%</i>	<i>±2.0%</i>	<i>13.8%</i>	<i>±0.9%</i>
Level 2 System software	OS	7.8%	±1.3%	6.6%	±1.2%	2.0%	±1.7%	5.8%	±1.2%	85.6%	±0.9%
	CGRTA	0.3%	±0.3%	0.3%	±0.3%	0%	—	0.1%	±0.1%	<0.1%	±0.0%
	<i>Subtotal</i>	<i>8.1%</i>	<i>±1.4%</i>	<i>6.9%</i>	<i>±1.2%</i>	<i>2.0%</i>	<i>±1.7%</i>	<i>5.9%</i>	<i>±1.2%</i>	<i>85.6%</i>	<i>±0.9%</i>
Level 3 Application level	Double exec.	—	—	—	—	—	—	1.7%	±0.6%	0.2%	±0.1%
	Checksum	—	—	—	—	0.4%	±0.8%	3.2%	±0.9%	0.1%	±0.1%
	<i>Subtotal</i>	<i>—</i>	<i>—</i>	<i>—</i>	<i>—</i>	<i>0.4%</i>	<i>±0.8%</i>	<i>4.8%</i>	<i>±1.1%</i>	<i>0.3%</i>	<i>±0.1%</i>
Other	Other unit	—	—	—	—	9.5%	±3.6%	6.1%	±1.2%	0.1%	±0.1%
	No error info.	10.3%	±1.5%	16.3%	±1.8%	0%	—	1.7%	±0.6%	0.2%	±0.1%
	<i>Subtotal</i>	<i>10.3%</i>	<i>±1.5%</i>	<i>16.3%</i>	<i>±1.8%</i>	<i>9.5%</i>	<i>±3.6%</i>	<i>7.8%</i>	<i>±1.4%</i>	<i>0.3%</i>	<i>±0.1%</i>
Fail-silence violations		1.3%	±0.6%	1.2%	±0.5%	1.2%	±1.3%	0%	—	0%	—
<i>Total number of errors</i>		<i>1549</i>		<i>1660</i>		<i>252</i>		<i>1513</i>		<i>6093</i>	

(a) Detection by the EDMs of the unit to which the faulted ICs belong

Error Detection Mechanisms		fault-injection with antennas									
		NOAMa		NOAMc		SEMCa		DEMCa(1)		DEMCa(2)	
		% (95% conf.)		% (95% conf.)		% (95% conf.)		% (95% conf.)		% (95% conf.)	
Level 1 Hardware	CPU	—	—	—	—	0%	—	0%	—	0%	—
	UEE	—	—	—	—	0%	—	0%	—	0%	—
	NMI	—	—	—	—	0%	—	4.0%	±1.0%	0.1%	±0.1%
	<i>Subtotal</i>	<i>—</i>	<i>—</i>	<i>—</i>	<i>—</i>	<i>0%</i>	<i>—</i>	<i>4.0%</i>	<i>±1.0%</i>	<i>0.1%</i>	<i>±0.1%</i>
Level 2 System software	OS	—	—	—	—	9.5%	±3.6%	2.1%	±0.7%	0%	—
	CGRTA	—	—	—	—	0%	—	0%	—	0%	—
	<i>Subtotal</i>	<i>—</i>	<i>—</i>	<i>—</i>	<i>—</i>	<i>9.5%</i>	<i>±3.6%</i>	<i>2.1%</i>	<i>±0.7%</i>	<i>0%</i>	<i>—</i>
Level 3 Application level	Double exec.	—	—	—	—	—	—	0%	—	0%	—
	Checksum	—	—	—	—	0%	—	0%	—	0%	—
	<i>Subtotal</i>	<i>—</i>	<i>—</i>	<i>—</i>	<i>—</i>	<i>0%</i>	<i>—</i>	<i>0%</i>	<i>—</i>	<i>0%</i>	<i>—</i>

(b) Detection by the EDMs of the other unit (detail of "Other unit" entry in Table (a) above)

Table 9: Results of EMI

Among the hardware EDMs, the CPU mechanisms strongly dominate (about 75% in most cases) followed by a much smaller amount of NMI EDMs (about 5%). Some UEEs were also observed in all campaigns.

The OS EDMs dominate for the software level mechanisms and the message checksum EDMs dominate for the application level mechanisms. Most of the errors detected by the OS EDMs in the DEMCa(2) campaign indicate that a message that was required by the application was lost.

The percentage of fail-silence violations was 1.3% for the NOAMa combination, 1.2% for the SEMCa (note however the large uncertainty in this figure as it is based on only 252 errors, i.e. the confidence interval is  $\pm 1.3\%$ ) and 0% for the DEMCa combinations (7606 errors were observed totally for this combination). There were never any significant differences regarding the percentage of fail-silence violations between the cases when the application unit was fault-injected (NOAMa), and when the fault injection was focused on the communication unit (NOAMc).

The supplementary detections delivered by the other fault free unit of the tested node were made by NMI and OS error detection mechanisms only.

## 6 Detailed Analysis of Experimental Results

The results presented in Chapter 5 are analysed in detail in this chapter, in which the three fault injection techniques are compared by studying the error detections made for each technique.

### 6.1 Comparison of Results

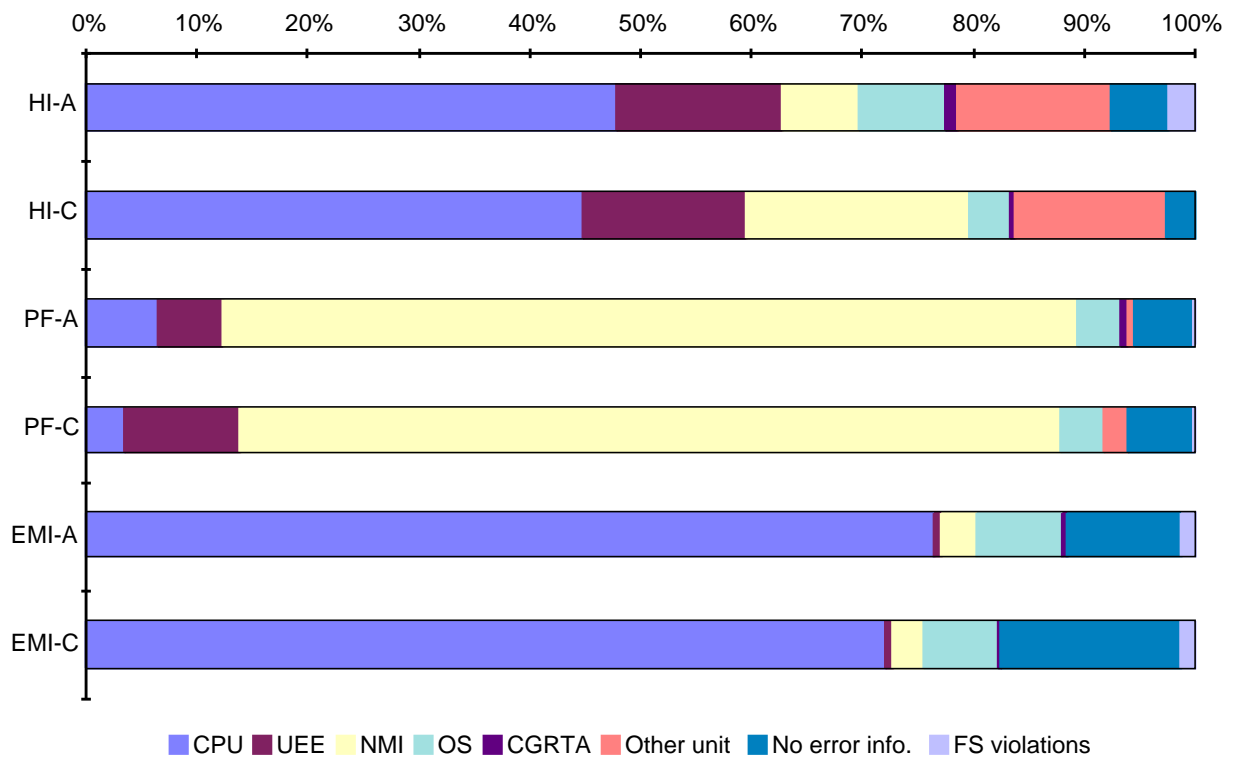
Chapter 5 showed that the hardware EDMs detected most of the errors for all fault injection campaigns, except for the EMI DEMCa(2) campaign in which the OS EDMs dominated. However, this campaign differs so radically from all other campaigns that it will not be included in the comparison in order to simplify the discussion.

The main difference between the fault injection techniques, when looking at the hardware EDMs, is that the CPU EDMs dominate for the heavy-ion radiation and EMI techniques, while the NMI EDMs dominate for the pin-level technique. A closer examination of the results (e.g. see Section 6.4) shows that heavy-ion radiation exercised seven of the eight CPU EDMs, while pin-forcing exercised six and EMI five of the CPU EDMs. The corresponding figures for the NMI EDMs are 12, 16 and 9, respectively. This indicates that the pin-forcing technique may be more effective than the other techniques in exercising hardware EDMs located outside the fault injected IC, while the heavy-ion technique may be more effective in exercising mechanisms within the fault injected IC. The percentage of UEEs is fairly large when using heavy-ion radiation (about 10% to 15%) and pin-forcing (about 5% to 10%), but quite small when using EMI (never larger than 3.2%).

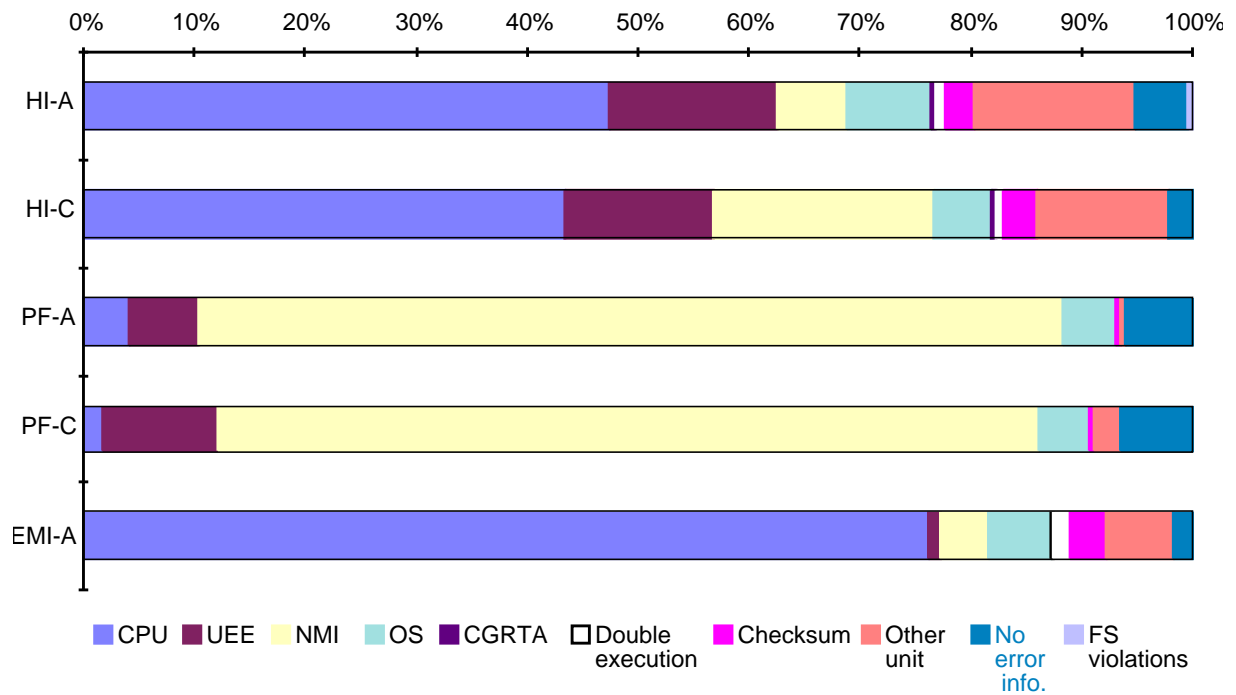
The software EDMs detected the second largest amount of errors for all techniques. Here the OS EDMs strongly dominate. This domination is larger for the pin-forcing and EMI techniques than for the heavy-ion radiation technique, but it is difficult to speculate on why the difference exist due to the lack of observability in the experiments.

While the application level EDMs detected the smallest amount of errors for all techniques, they were nevertheless necessary since the fail-silence coverage was significantly reduced when these were disabled. Here the message checksum EDMs dominate for all techniques. The double execution mechanism was never exercised by the pin-forcing technique.

The results also show that the heavy-ion radiation technique stressed the system the most as it caused the most fail-silence violations of all techniques. This technique also generated the largest error set, as indicated by the spread of the error detections among the EDMs. The spread of the error detections among the categories given in Section 5.2 is larger for the EMI technique than for the pin-forcing technique, but a more detailed comparison (see Sections 6.2 and 6.4) shows that the pin-forcing technique achieved a larger spread of the error detections among the specific EDMs within the exercised categories.



**Figure 14: Results obtained using NOAM combinations**



**Figure 15: Results obtained using DEMC combinations**



A summary of a representative selection of results is given in Figures 14 and 15, where the differences between the techniques are clearly seen. Both figures show the observed distributions of the error detections among the error detection mechanisms for all the categories given in Section 5.2. Figure 14 shows the results obtained using each technique when the application level EDMs were deactivated (i.e. the NOAM combinations), and Figure 15 shows the corresponding results when all the EDMs were activated (i.e. the DEMC combinations). The acronyms on the left side of the diagrams indicate which fault injection technique was used: HI in the case of the heavy-ion radiation technique, PF where pin-level fault injection using forcing was used and EMI for the EMI technique using antenna wires and conducting plates. An ‘-A’ after the acronym indicates that the fault injection was focused on the application unit CPU, and a ‘-C’ that the fault injection was focused on the communication unit CPU. The figures show that the CPU EDMs and OS EDMs always dominate more on the application unit than on the communication unit. While the primary reason for these differences may be that the workload is quite different between the two units, the hardware dissimilarities between the units may also add to the differences. Note, however, that the differences between the *techniques* are much greater than the differences that exist between the two units when looking at a *single* fault injection technique.

## 6.2 Detailed Comparison

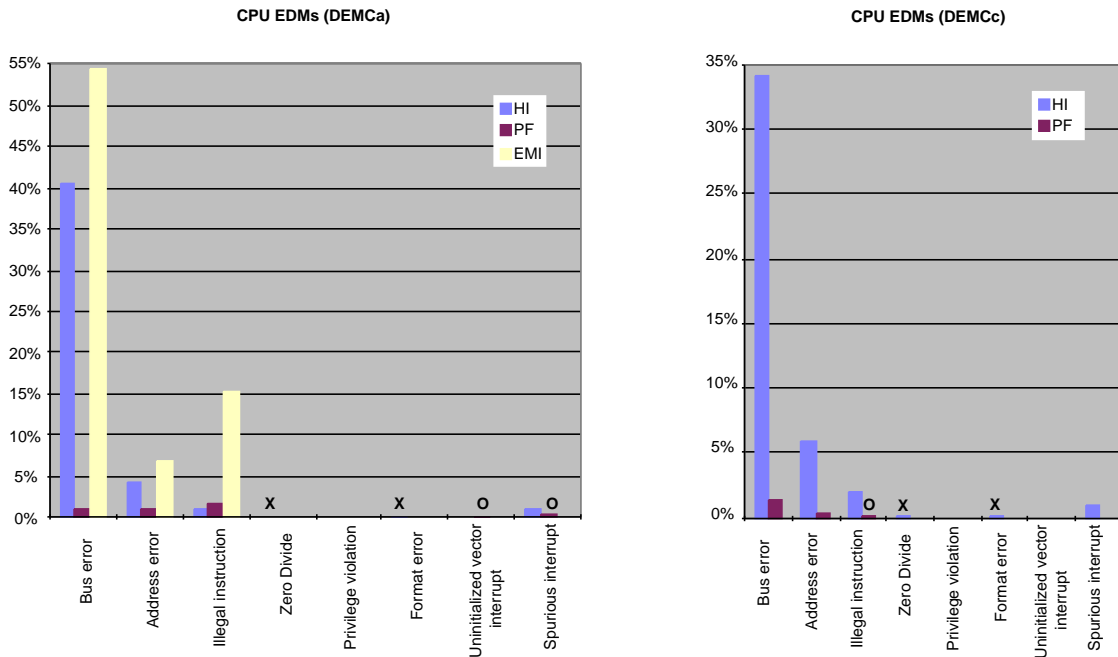
The distribution of the errors among the various EDMs for the three physical fault injection techniques are described in detail and compared in this section. The combinations considered are DEMCa in the case of heavy-ion radiation, pin-forcing and EMI, and DEMCc in the case of heavy-ion radiation and pin-forcing.

The DEMC combinations were chosen because they represent a fully equipped MARS set-up. For EMI, the DEMCa(1) combination was chosen rather than the DEMCa(2) combination, since the results obtained using DEMCa(2) were so radically different from the other EMI combinations. The numbers of errors collected were 9036, 999 and 1531 for the heavy-ion radiation, pin-forcing and EMI DEMCa campaigns, respectively, and were 2437 and 1162 for the heavy-ion radiation and pin-forcing DEMCc campaigns, respectively.

The results obtained from these campaigns are shown in diagrams where the bars indicate the observed relative frequency of errors detected by the EDM out of the total number of observed errors in each campaign. The symbols  $\times$ ,  $\circ$  and  $\Delta$  indicate that the percentage of detections made by the heavy-ion radiation, pin-forcing and EMI techniques, respectively, for this particular EDM, was very low and therefore hard to visualize in the diagrams.

### 6.2.1 Distribution of Errors Among CPU EDMs

Figure 15 shows the distribution of the errors among the eight CPU EDMs for the DEMCa campaigns (left) and DEMCc campaigns (right).



**Figure 16: Distribution of errors among the CPU EDMs**

The “Bus error”, “Address error” and “Illegal instruction” EDMs dominate for all techniques. This is not surprising since the address and data buses are used frequently and are connected to the majority of the pins of the CPU. Among the other CPU EDMs, only “Spurious interrupts” occurred with any major frequency (for the heavy-ion radiation technique).

No major differences between the DEMCa and DEMCc combinations exist for the heavy-ion radiation technique, except for a lower number of “Bus errors” for DEMCc. For the pin-forcing technique, the amounts of “Bus errors”, “Address errors” and “Illegal instructions” are reversed between the DEMCa and DEMCc combinations so that the distribution for the DEMCc combination is similar to the heavy-ion distribution.

Heavy-ion radiation exercised the most CPU EDMs (six different kinds), followed by pin-forcing (five) and EMI (three).

### 6.2.2 Distribution of Errors Among NMI EDMs

The distribution of the errors among the 16 NMI EDMs is shown in Figure 17. On some rare occasions, the experimental set-up did not deliver any information about which specific NMI mechanism detected the error, just that an NMI had been activated. These cases have been categorized as “Unknown” in the diagrams. These error reports may have been caused by some internal mechanism activating the NMI line of the processor (or “spikes” in the NMI line), i.e. no NMI hardware was involved since the input latches described in Section 2.5.1 must have been unaffected.

The letters ‘A’ and ‘C’ before the name of each EDM denote whether the EDM is built into the application unit or communication unit of the tested node.

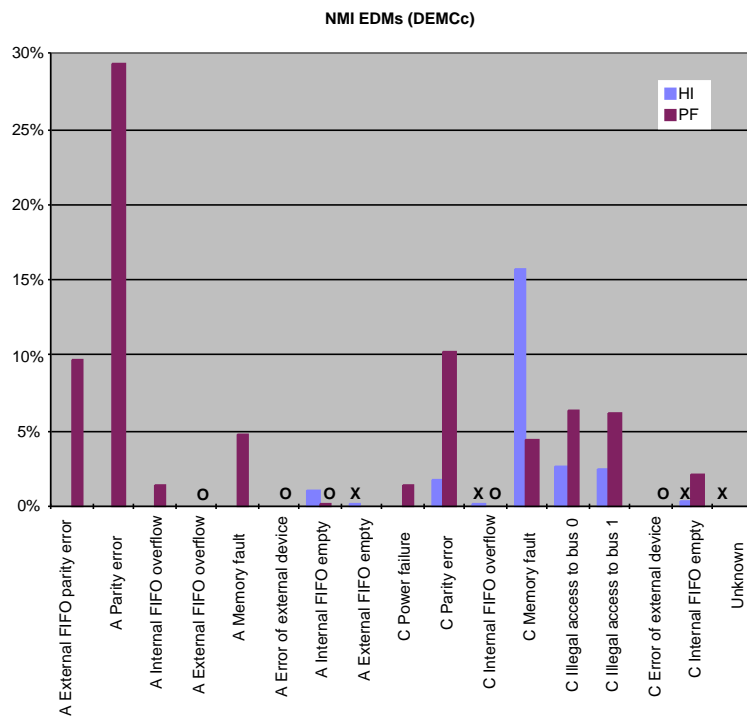
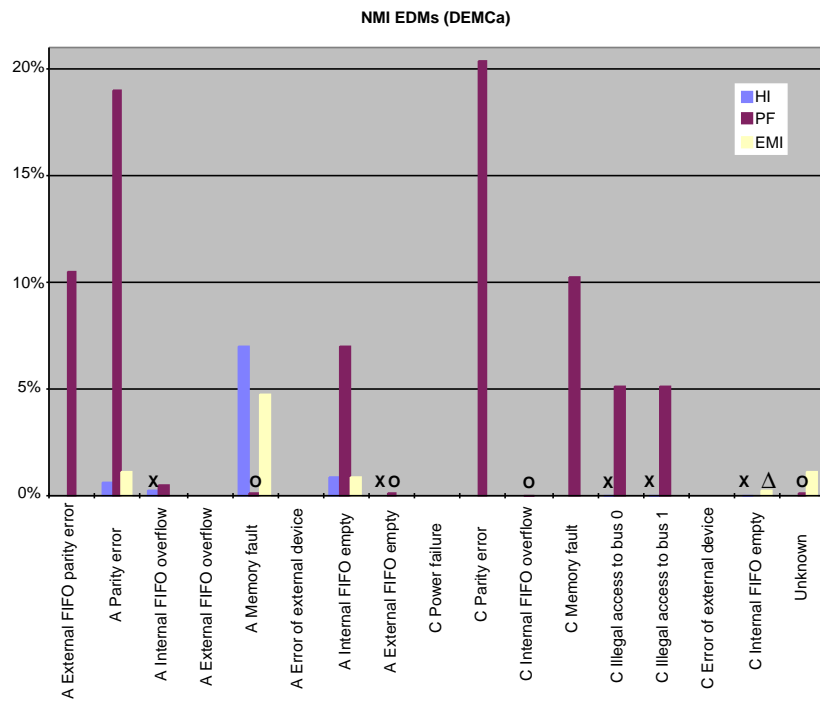
When two or more NMI EDMs were triggered at the same time (see Section 2.5.1), the effect from each triggered NMI EDM has been added equally to the proportions shown in the diagrams, i.e. the amount added was obtained by dividing the effect of one error with the number of NMI EDMs that were triggered for this error. E.g. if the NMIs that were triggered for an error were “A Memory Fault” and “A Parity Error” (i.e. two NMIs were triggered for one error), the “A Memory Fault” and “A Parity Error” NMIs counted as 0.5 errors each.

The most common NMI error detections are “Parity errors” and “Memory faults”. “Parity errors” dominate for pin-forcing, while “Memory faults” dominate for heavy-ion radiation and EMI. (The reason for this may lie in the choice of circuits used for pin-forcing.) A fair amount of “Internal FIFO empty” errors occurred for all techniques as well.

For pin-forcing and heavy-ion radiation, the Time Slice Controller, which prevents access to the MARS bus at an illegal point in time, detected several errors. These are the “Illegal access to bus 0” and “Illegal access to bus 1” NMIs. For DEMCa, these mechanisms detected 0.1% of all errors for the heavy-ion radiation technique and 10.27% for pin-forcing. For DEMCc, the corresponding figures are 5.04% and 12.46%. Without this mechanism, the fail-silence property would have been violated in the time domain, which could have lead to a system failure. No such fail-silence violations were observed in any of these campaigns.

The differences between the two units are more apparent for the NMI EDMs than for the CPU EDMs. When comparing the DEMCa and DEMCc campaigns, both heavy-ion radiation and pin-forcing show fairly different distributions. These differences are also apparent even when the fact that more NMI EDMs should belong to the fault injected unit is considered. Memory faults more than doubled for the DEMCc campaign when heavy-ion radiation was used. For pin-forcing, the amounts of “A Parity errors” and “C Parity errors” are similar for the DEMCa campaign, while the number of “A Parity errors” is nearly three times that of “C Parity errors” for the DEMCc campaign. The different workload running at each unit may be the primary cause of these differences, but some part may be due to the dissimilarities in hardware that exist between the units.

Pin-forcing managed to exercise all 16 NMI EDMs, while the corresponding figures for the heavy-ion radiation and EMI techniques are 11 and four, respectively, for the two campaigns considered here.



**Figure 17: Distribution of errors among NMI EDMs**

### 6.2.3 Distribution of Errors Among System Software EDMs

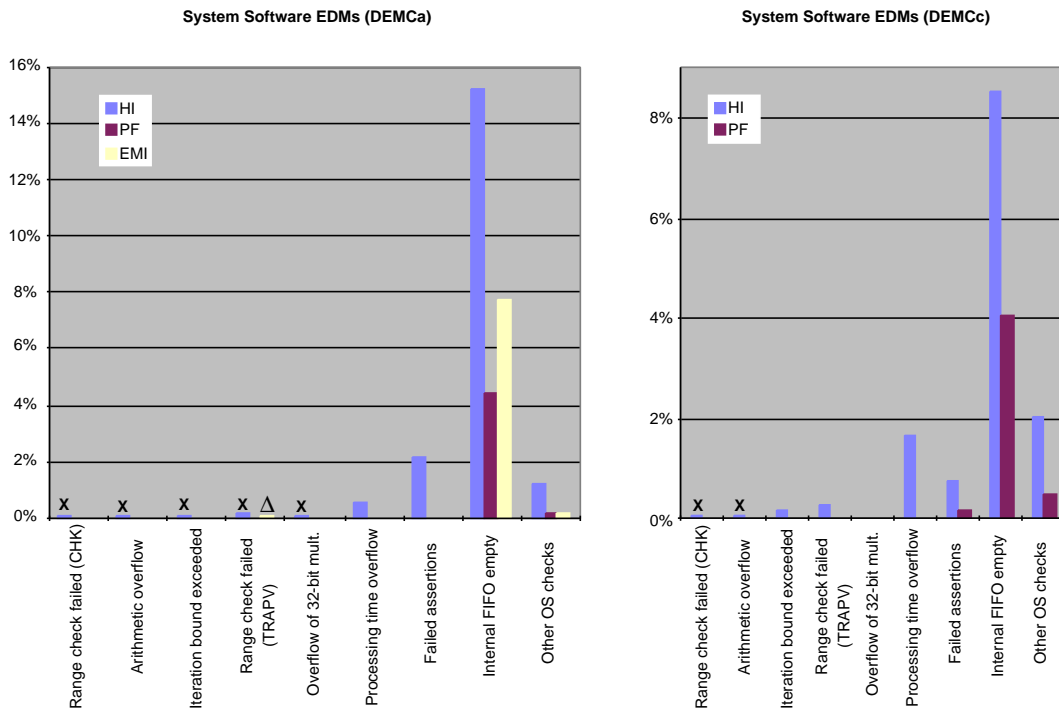
The system software EDMs have been grouped into the following nine categories as shown in Figure 18 (the total number of system software EDMs is 263):

The CGRTA mechanisms:

- Value range overflow: “Range check failed (CHK)”, “Arithmetic overflow”, “Range check failed (TRAPV)” and “Overflow of 32-bit multiplication”.
- Loop iteration bound overflow: “Iteration bound exceeded”.

EDMs built into the operating system:

- Processing time overflow: “Processing time overflow”.
- Various checks on OS data: “Internal FIFO empty” (4 EDMs) and “Other OS checks” (79 EDMs).
- Various assertions in the OS: “Failed assertions” (169 EDMs).



**Figure 18: Distribution of errors among system software EDMs**

The “Internal FIFO empty” mechanism is by far the most dominant system software mechanism for all techniques. This mechanism is activated when one of the units of the node tries to read a message which has not been delivered by the other unit, e.g. when the other unit has failed. This error was often reported by the fault free unit of the tested node when the fault-injected unit failed to report any other error information. Among the CGRTA EDMs, the “Range check failed (TRAPV)” mechanism dominates. It was exer-

cised by both heavy-ion radiation and EMI, while no CGRTA mechanisms at all were exercised by pin-forcing. The proportion of CGRTAs is much lower than the OS EDMs for both the heavy-ion radiation and EMI techniques.

Some differences between the DEMCa and DEMCc campaigns can be seen. Fewer “Failed assertions” were observed for heavy-ion radiation for the DEMCc combination, while the reverse is true for pin-forcing. The amount of “Processing time overflow” EDMs nearly tripled for heavy-ion radiation for the DEMCc combination. No “Overflow of 32-bit multiplications” were observed for the DEMCc campaigns.

Heavy-ion radiation managed to exercise mechanisms of all nine categories, while pin-forcing and EMI exercised mechanisms of only three categories each.

### 6.2.4 Distribution of Errors Among Other Mechanisms

Figure 19 shows the distribution of the errors not included in Sections 6.2.1-6.2.3. For heavy-ion radiation and pin-forcing, the “Unexpected exceptions” clearly dominate and for EMI, the “Message checksum errors” had been exercised the most. No “Double execution errors” at all were exercised by pin-forcing.

All techniques caused a substantial amount of errors lacking any error information (the “No error information” category). Fail-silence violations were observed only for the heavy-ion radiation technique.

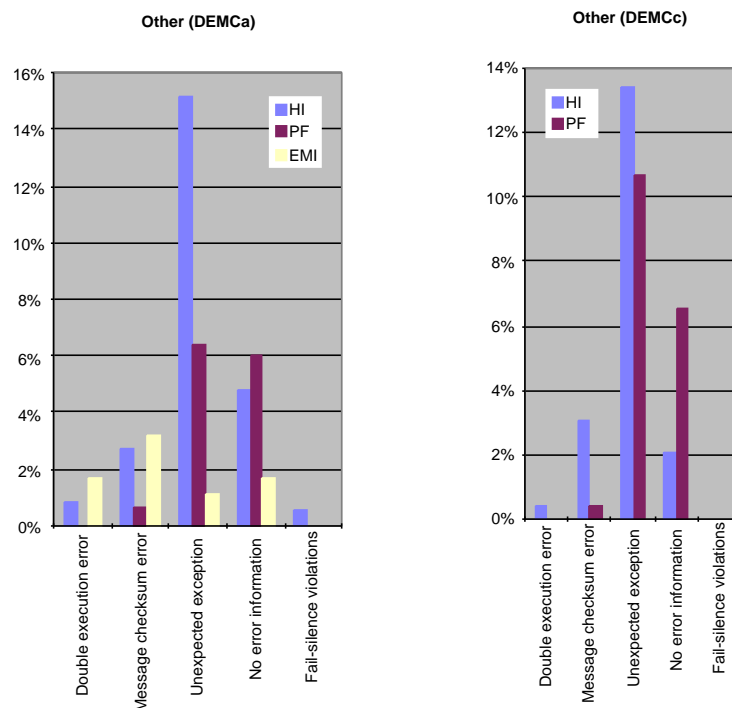


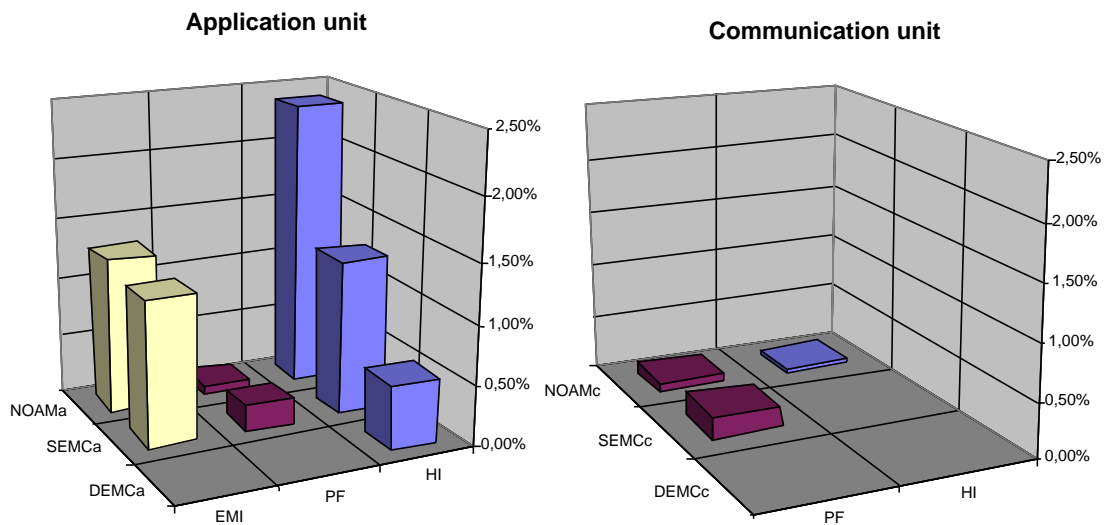
Figure 19: Distribution of errors among other mechanisms

No major differences exist between the DEMCa and DEMCc campaigns except that the percentage of “No error information” errors is substantially lower for DEMCc when using heavy-ion radiation.

The heavy-ion radiation technique was the only technique that caused any fail-silence violations when the DEMC combination was used (0.5% for DEMCa). No fail-silence violations occurred in the DEMCc campaigns.

### 6.3 Fail-Silence Violations Observed

The most important feature of the MARS system studied here, namely the fail-silence property of a MARS node, is analysed in this section. Figure 20 shows the percentage of fail-silence violations when fault injection was focused on the application unit CPU (left) and communication unit CPU (right) for the NOAM, SEMC and DEMC combinations. The absence of a bar indicate that no fail-silence violations were observed for this combination.



**Figure 20: Fail-silence violations observed**

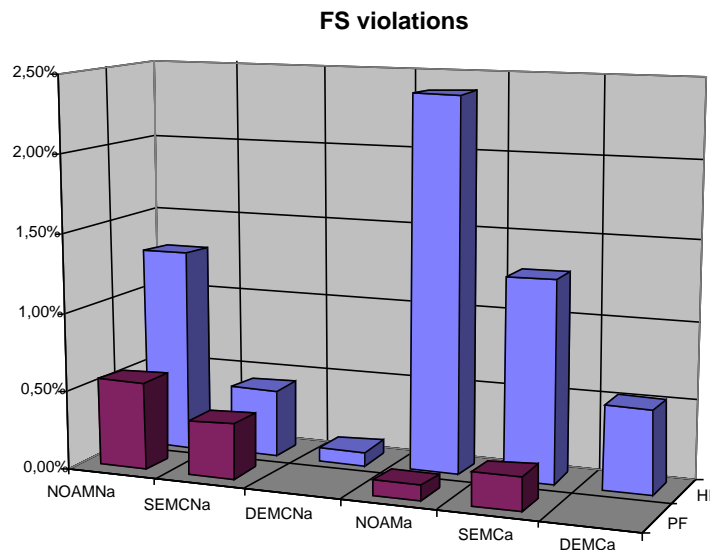
The results show that the application unit is a great deal more sensitive to faults than the communication unit, as more fail-silence violations occurred when fault injecting the application unit (at least when the fault injection is focused to the CPUs). This can be explained by the fact that the result is being produced by the application unit and is merely transferred to the MARS bus by the communication unit. The amount of time spent on manipulation of vulnerable data by the fault injected IC will, of course, affect the probability of the data being erroneous, i.e. the activity of the system has a high impact on the coverage factor.

The percentage of fail-silence violations was largest for heavy-ion radiation. Several fail-

silence violations also occurred for the EMI technique, while the pin-forcing technique produced substantially fewer fail-silence violations.

When all EDMs were used (DEMC), the percentage of fail-silence violations was smallest. Only the heavy-ion radiation technique managed to cause fail-silence violations for this combination (0.5% for the DEMCa combination). The percentage of fail-silence violations was largest when no application level EDMs were used (NOAM), except for pin-forcing, which shows a slight overrepresentation of fail-silence violations for the SEMC combination (it should be noted, however, that the confidence intervals are relatively large for pin-forcing due to the relatively low number of errors collected).

Figure 21 shows the percentage of fail-silence violations observed for pin-forcing and heavy-ion radiation for all combinations when the application unit was fault-injected.



**Figure 21: Fail-silence violations on the application unit**

The results show that the application level EDMs were more efficient for improving the fail-silence coverage than the NMI mechanisms, as more fail-silence violations were observed when switching off the application level EDMs than when the NMIs were deactivated. This can be seen when comparing the DEMCNa and NOAMa combinations with the DEMCa combinations for each technique.

The anomaly discussed in Section 5.3 can also be clearly seen in Figure 21. Fewer fail-silence violations were observed when the NMIs were deactivated than when they were activated for the heavy-ion radiation technique. For pin-forcing, however, the MARS node behave as expected, i.e. there were more fail-silence violations when more EDMs were switched off. The reason for the anomaly may lie in the design of the heavy-ion radiation technique, as the campaigns for all combinations with the NMIs switched off were generally conducted after the other campaigns, when the target IC had been exposed to heavy-ion radiation longer (i.e. the IC was more contaminated), the pressure in the miniature



vacuum chamber was generally higher (i.e. the chamber had not been vacuum-pumped recently), and the same anomalous Cf-252 source had been used longer (i.e. the activity of the source was lower and its energy spectrum may have been different since it was either badly manufactured or had accidentally been touched, see Section 5.3). The amount of fail-silence violations observed in the campaigns made using a new sample of the irradiated CPU, but the same Cf-252 source, was lower than one half of that observed in the original campaigns (see Section 5.3). Whether this was caused by differences in the energy spectrum and activity of the Cf-252 source or by the change of target CPU, remains to be investigated. Clearly, more research is needed to examine the heavy-ion radiation technique further concerning these parameters.

## 6.4 Error Detection Mechanisms Exercised

Table 10 summarizes the detections made for each technique when fault injecting the application unit (left side of the table) and communication unit (right side). An **X** in the table cell indicates that one or more detections were made by the EDM for the technique considered. This summary is based on all campaigns made for heavy-ion radiation and pin-forcing, and most of the campaigns made for EMI (on the application unit only).

Heavy-ion radiation managed to exercise seven CPU mechanisms (all CPU mechanisms except the “Uninitialized vector interrupt”), while pin-forcing managed six mechanisms and EMI five mechanisms. No major differences exist between the application unit and communication unit. Fewer types of CPU EDMs were exercised by pin-forcing on the communication unit than on the application unit, but this can be explained by the fact that fewer errors were collected on the communication unit.

Pin-forcing managed to exercise all 16 NMI mechanisms, while heavy-ion radiation managed to exercise 12 mechanisms and EMI 9 mechanisms. The differences between the two units are most prominent for the heavy-ion radiation technique, which clearly shows that the NMI EDMs belonging to the fault injected unit were exercised more than the NMI EDMs belonging to the fault free unit. For pin-forcing, the errors propagated to the fault free unit more easily, as there is no visible correspondence between which NMI EDMs were exercised and the unit that was fault-injected.

All fault injection techniques managed to exercise the TSC EDMs (the “Illegal access to bus 0” and “Illegal access to bus 1” EDMs), and no fail-silence violations were observed in the time domain in any of the campaigns when these NMI EDMs were activated. This indicates that the TSC actively prevents fail-silence violations in the time domain.

All hardware mechanisms were exercised by at least one of the techniques.

The system software EDMs have been grouped into the categories described in Section 6.2.3. The heavy-ion radiation managed to exercise EDMs in all of the nine categories, while pin-forcing managed five and EMI three. A look at the results in more detail reveals that 44 of the 263 system software EDMs were exercised by heavy-ion radiation, while 16 mechanisms were exercised by pin-forcing and seven by EMI. The mechanisms that

---

were not exercised by any technique were mainly scheduling errors (except “Processing time overflow”) and failed assertions in the protocol software.

Pin-forcing is the only technique that was unable to exercise any “Double execution errors”. All techniques triggered a substantial amount of “Unexpected exceptions” and “No error information” errors. These are errors that the MARS node designers had not anticipated. The MARS node also managed to deliver incorrect results for many of the combinations considered in Section 5.1, according to each physical fault injection technique, as indicated by the number of fail-silence violations obtained.

		HI-A	PF-A	EMI-A	HI-C	PF-C
<b>CPU Error Detection Mechanisms</b>						
Bus error		X	X	X	X	X
Address error		X	X	X	X	X
Illegal instruction		X	X	X	X	X
Zero divide		X		X	X	
Privilege violation		X	X		X	
Format error		X			X	
Uninitialized vector interrupt			X			
Spurious interrupt		X	X	X	X	X
<b>NMI Error Detection Mechanisms</b>						
Application unit	External FIFO parity error		X			X
	Parity error	X	X	X	X	X
	Internal FIFO overflow	X	X	X		X
	External FIFO overflow	X	X			X
	Memory fault	X	X	X	X	X
	Error of external device					X
	Internal FIFO empty	X	X	X	X	X
	External FIFO empty	X	X		X	
Communication unit	Power failure		X			X
	Parity error		X	X	X	X
	Internal FIFO overflow		X	X	X	X
	Memory fault	X	X		X	X
	Illegal access to bus 0		X	X	X	X
	Illegal access to bus 1		X	X	X	X
	Error of external device		X			X
	Internal FIFO empty	X	X	X	X	X
<b>System Software Error Detection Mechanisms</b>						
Range check failed (CHK)		X			X	
Arithmetic overflow		X			X	
Iteration bound exceeded		X			X	
Range check failed (TRAPV)		X	X	X	X	
Overflow of 32-bit multiplication		X				
Processing time overflow		X	X		X	
Failed assertion		X	X		X	X
Internal FIFO empty		X	X	X	X	X
Other OS checks		X	X	X	X	X
<b>Other Error Detection Mechanisms</b>						
Double execution error		X		X	X	
Message checksum error		X	X	X	X	X
Unexpected exceptions		X	X	X	X	X
No error information		X	X	X	X	X
Fail-silence violations		X	X	X	X	X
Total number of errors		32008	4587	25483	13885	3273

Table 10: Error detection mechanisms exercised

## 7 Conclusions

Three physical fault injection techniques—heavy-ion radiation, pin-forcing, and EMI—were applied on the fault-tolerant, distributed, real-time system MARS in a unique study aimed at comparing the techniques and validating the MARS system.

The techniques were compared both in a general manner in which they were compared according to five attributes (controllability, flexibility, repeatability, physical reachability and timing measurement) and by fault injection experiments using the same MARS system set-up at three different sites with a different fault injection technique applied at each site.

The general comparison of the fault injection techniques shows that pin-level and EMI fault injection are more flexible than the heavy-ion radiation technique. The pin-level technique has a much higher controllability than both the heavy-ion radiation and EMI techniques, although the heavy-ion radiation technique has the unique feature of being able to inject faults internally in integrated circuits. Only pin-level fault injection has a high ability to acquire timing information about monitored events (e.g. measuring the error detection latency) of the three techniques used in the study.

The results of the fault injection experiments show fairly large differences in the distribution of the error detections among the various EDMs for the three physical fault injection techniques. This suggests that the techniques are somewhat complementary, i.e. they generate fairly different error sets. The pin-forcing technique most exercised the hardware EDMs located outside the CPU, while the heavy-ion radiation and EMI techniques appear to be more suitable for exercising application level EDMs. The only technique that exercised a fairly diverse set of system software EDMs was heavy-ion radiation. This technique also stressed the system most, as it caused the most fail-silence violations, and it was the only technique that caused any fail-silence violations when all EDMs were activated in the MARS system. The heavy-ion radiation technique also showed the largest spread in the detections among the EDMs, and was the most effective technique in exercising mechanisms located within the fault-injected CPU. The errors detectable by NMI EDMs propagated between the two units of the tested node most easily using the pin-forcing technique.

Some difficulties were encountered for each technique. For the heavy-ion radiation technique anomalous results were obtained when comparing the number of fail-silence violations observed in campaigns with and without activated NMI EDMs. For pin-forcing, there was comparatively low stress on the tested node as well as very few system software and application level EDMs exercised and, for EMI, severe difficulties in statistically reproducing the results were encountered.

All techniques managed to cause a substantial proportion of errors that the MARS designers had not anticipated (i.e. “No error information” and UEE errors) and were particularly effective in exercising hardware EDMs (all hardware EDMs built into the MARS node were exercised). Several system software EDMs were not exercised by any of the tech-

niques. Future research using software-implemented fault injection (SWIFI) and other types of fault injection methods will show the validity of using these mechanisms.

The hardware EDMs detected most of the errors generated by the three fault injection techniques. This may be due to the nature of the techniques, i.e. they all affect the hardware directly.

The results also show that the application level EDMs are necessary for improving fail-silence coverage. Fail-silence violations were observed for all three techniques when the application level EDMs were not used, and were observed for heavy-ion radiation only—and in much lower quantities—when these EDMs were used. They were also among the most effective EDMs for improving fail-silence coverage. This is particularly noticeable when comparing the results for the combinations in which only the NMI EDMs were deactivated with the combinations in which only the application level EDMs were deactivated. More fail-silence violations were observed when the application level EDMs were deactivated than when the NMI EDMs were deactivated.

The Time Slice Controller was very effective in preventing fail-silence violations in the time domain. No such fail-silence violations were observed when this mechanism was used. When it was switched off, three possible fail-silence violations in the time domain were observed (for the NOAMNc combination using heavy-ion radiation).

The activity of the system was also shown to have a large impact on the fail-silence coverage, as more fail-silence violations always occurred when fault injecting the application unit than when the communication unit was fault injected. One explanation for this behaviour is that the result is produced by the application unit and is merely transferred to the MARS bus by the communication unit. The amount of time spent on manipulating vulnerable data by the fault injected ICs affects the probability of that data being erroneous. The hardware differences between the units may also have contributed the fail-silence coverage figures.

Some other effects on how the differences between the units impacted the coverage of the EDMs were shown. The CPU EDMs and OS EDMs dominate more on the application unit than on the communication unit for all techniques. Several other differences between the two units of the tested node exist when looking at each technique separately, particularly among the NMI EDMs and system software EDMs.

The results point to several interesting questions that future research should attempt to answer. What is the cause of the anomaly observed for heavy-ion radiation regarding the higher percentage of fail-silence violations when the NMI EDMs were not activated than when the NMIs were in use? Will another radiation source, showing a different energy spectrum, yield different results? What is the impact of the pressure, temperature, radiation and irradiation time on the heavy-ion radiation technique?

How do these techniques compare with e.g. SWIFI or fault injection via boundary and internal scan chains? It is still not clear whether hardware-implemented fault injection injects faults into a target system more closely representing the actual system than is the case when SWIFI is used, but hardware-implemented fault injection does at least produce

---

real hardware faults, similar to those that can occur in reality. A comparison between hardware-implemented fault injection and SWIFI is currently being made, as experiments using SWIFI have been conducted on a set-up similar to the one used in this study [Fuchs 1996].

The vital question of why fail-silence violations occurred for the heavy-ion radiation technique when all EDMs were activated in the MARS system could not be answered in this study since the development of a set-up for increasing the observability using a logic analyser connected to a comparator card was unsuccessful. Also, reproducibility for EMI was very low in these experiments. Is it possible to increase the reproducibility by further refining the EMI probe technique? The questions of how much of the complete fault/activity space was exploited by each physical fault injection technique and the workload used, and the overlap between the fault/activity subspaces created by these techniques and by reality also remain largely unanswered.

## References

- [Arlat *et al.* 1990] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J.-C. Fabre, J.-C. Laprie, E. Martins and D. Powell, "Fault Injection for Dependability Validation — A Methodology and Some Applications", *IEEE Transactions on Software Engineering*, 16 (2), pp.166-82, February 1990.
- [Damm 1986] A. Damm, "The Effectiveness of Software Error-Detection Mechanisms in Real-Time Operating Systems", in *Proc. 16th Int. Symp. on Fault-Tolerant Computing (FTCS-16)*, Vienna, Austria, pp.171-6, IEEE Computer Society, 1986.
- [Damm 1988] A. Damm, *Experimental Evaluation of Error-detection and Self-Checking Coverage of Components of a Distributed Real-time System*, Doctoral Dissertation, Technical University, Vienna, Austria, October 1988.
- [Elder *et al.* 1988] J. Elder, J. Osborn, W. Kolasinski, R. Koga, "Method for Characterizing a Microprocessor's Vulnerability to SEU", *IEEE Transactions on Nuclear Science*, 35 (6), pp.1678-81, 1988.
- [Fuchs 1996] E. Fuchs, "An Appraisal of the Error Detection Mechanisms in MARS using Software Implemented Fault Injection", in *Proc. 2nd European Dependable Computing Conference (EDCC-2)*, Taormina, Italy, October 1996.
- [Grünsteidl, Kantz and Kopetz 1991] G. Grünsteidl, H. Kantz and H. Kopetz. "Communication Reliability in Distributed Real-Time Systems," in *10th IFAC Workshop on Distributed Computer Control Systems*, Semmering, Austria, Sept. 1991.
- [Grünsteidl and Kopetz 1991] G. Grünsteidl and H. Kopetz, "A Reliable Multicast Protocol for Distributed Real-Time Systems", in *8th IEEE Workshop on Real-Time Operating Systems and Software*, (Atlanta, GA, USA), IEEE, 1991.
- [Gunnflo *et al.* 1987] U. Gunnflo, J-B. Johnsson, J. Karlsson, S. Loeb and J. Torin, *A Fault Injection System for the Study of Transient Fault Effects on Computer Systems*, Tech. Report No. 47, Dept. of Comp. Eng., Chalmers University of Technology, Göteborg, Sweden, 1987.
- [Iyer and Tang 1993] R. K. Iyer and D. Tang. *Experimental Analysis of Computer System Dependability*, Tech. Report CRHC-93-15, CRHC, Coord. Science Lab., University of Illinois at Urbana-Champaign, Urbana, Illinois, USA, 1993.
- [Johansson 1993] R. Johansson, *On Single Event Phenomena in Microprocessors*, Tech. Report No. 162L, Dept. of Comp. Eng., Chalmers University of Technology, Göteborg, Sweden, 1993.
- [Karlsson *et al.* 1994] J. Karlsson, P. Lidén, P. Dahlgren, R. Johansson and U. Gunnflo, "Using Heavy-Ion Radiation to Validate Fault-Handling Mechanisms", *IEEE Micro*, 14 (1), pp.8-23, February 1994.
- [Karlsson *et al.* 1995] J. Karlsson, P. Folkesson, J. Arlat, Y. Crouzet, G. Leber, J. Reis-

- inger, "Application of Three Physical Fault Injection Techniques to the Experimental Assessment of the MARS Architecture", in *Proc. Fifth Int. Working Conf. on Dependable Computing for Critical Applications (DCCA-5)*, (Urbana-Champaign, IL, USA), pp.150-161, Sept. 1995.
- [Kopetz and Ochsenreiter 1987] H. Kopetz and W. Ochsenreiter, "Clock Synchronization in Distributed Real-Time Systems", *IEEE Transactions on Computers*, 36 (8), pp.933-940, August 1987.
- [Kopetz *et al.* 1989] H. Kopetz, A. Damm, C. Koza, M. Mulazzani, W. Schwabl, C. Senft and R. Zainlinger, "Distributed Fault-Tolerant Real-Time Systems: The MARS Approach", *IEEE Micro*, 9 (1), pp.25-40, February 1989.
- [Kopetz *et al.* 1991] H. Kopetz, P. Holzer, G. Leber and M. Schindler. *The Rolling Ball on MARS*, Research Report 13/91, Institut für Technische Informatik, Technische Universität Wien, Vienna, Austria, 1991.
- [Krüger and Nossal 1993] A. Krüger and R. Nossal, *The MARS Programming Guide Version 1.0*, 1993.
- [Lala 1983] J. H. Lala, "Fault Detection, Isolation, and Reconfiguration in FTMP: Methods and Experimental Results", in *Fifth AIAA/IEEE Digital Avionics Sys. Conf.*, pp.21.3.1-.3.9, 1983.
- [Laprie 1985] J.-C. Laprie, "Dependable Computing and Fault Tolerance: Concepts and Terminology", in *Proc. 24th Int. Symp. on Fault-Tolerant Computing (FTCS-24)*, pp.2-11, (Ann Arbor, MI, USA), June 1985.
- [Lidén *et al.* 1994] P. Lidén, P. Dahlgren, R. Johansson, J. Karlsson, "On Latching Probability of Particle Induced Transients in Combinational Networks", in *Proc. 24th Int. Symp. on Fault-Tolerant Computing (FTCS-24)*, pp.340-49, (Austin, TX, USA) June 1994.
- [Madeira *et al.* 1994] H. Madeira, M. Rela, F. Moreira and J. G. Silva, "A General Purpose Pin-level Fault Injector", in *Proc. 1st European Dependable Computing Conf. (EDCC-1)*, (Berlin, Germany), pp.199-216, Springer-Verlag, 1994.
- [Philips Semiconductors 1991] Philips Semiconductors, *SCC68070 User Manual 1991, Part 1 - Hardware*, 1992.
- [Powell *et al.* 1988] D. Powell, G. Bonn, D. Seaton, P. Veríssimo and F. Waeselynck, "The Delta-4 Approach to Dependability in Open Distributed Computing Systems", in *Proc. 18th Int. Symp. on Fault-Tolerant Computing Systems (FTCS-18)*, (Tokyo, Japan), pp.246-51, IEEE Computer Society Press, 1988.
- [Powell *et al.* 1995] D. Powell, E. Martins, J. Arlat and Y. Crouzet, "Estimators for Fault Tolerance Coverage Evaluation", *IEEE Transactions on Computers*, 44 (2), pp.261-74, February 1995.
- [Reisinger 1993] J. Reisinger, *MARS Operating System User Manual Version 2.0*, 1993.
-



- [Reisinger *et al.* 1995] J. Reisinger, A. Steininger, G. Leber, “The PDCS Implementation of MARS Hardware and Software”, in *ESPRIT Basic Research Series: Predictably Dependable Computing Systems*, pp.209-24, Springer Verlag, 1995.
- [Schuette *et al.* 1986] M. A. Schuette, J. P. Shen, D. P. Siewiorek and Y. X. Zhu, “Experimental Evaluation of Two Concurrent Error Detection Schemes”, in *Proc. 16th Int. Symp. Fault-Tolerant Computing (FTCS-16)*, (Vienna, Austria), pp.138-43, IEEE Computer Society Press, 1986.
- [Sokol *et al.* 1987] J. Sokol, W. Kolasinski, M. Wong, R. Koga, R. Suhrke, T. Frey, “Advantage of Advanced CMOS Over Advanced TTL in a Cosmic Ray Environment”, *IEEE Transactions on Nuclear Science*, 34 (6), pp.1338-40, December 1987.
- [Steininger and Reisinger 1993] A. Steininger and J. Reisinger, *Dual Processor Mars-Node Hardware Documentation Version 2.2*, 8 March 1993.
- [Vrchoticky 1992] A. Vrchoticky. *Modula/R Language Definition*, Research Report 2/92, Institut für Technische Informatik, Technische Universität Wien, Vienna, Austria, 1992.
- [Walter 1990] C. J. Walter, “Evaluation and Design of an Ultra-Reliable Distributed Architecture for Fault Tolerance”, *IEEE Transactions on Reliability*, 39 (4), pp.492-9, October 1990.

## Appendix A: Acronyms used

ADF	Application Definition File
CGRTA	Compiler Generated Run-Time Assertion
CMOS	Complementary Metal Oxide Semiconductor
CPU	Central Processing Unit
CS	Cold Start
CSU	Clock Synchronization Unit
DEMC	Double Execution Message Checksum
DEMCN	Double Execution Message Checksum, No NMIs
DMA	Direct Memory Access
DRAM	Dynamic Random Access Memory
EDM	Error Detection Mechanism
EMI	ElectroMagnetic Interference
EPROM	Erasable Programmable Read Only Memory
FIFO	First In First Out
FTU	Fault-Tolerant Unit
HI	Heavy-Ion radiation
I <sup>2</sup> C	Inter-Integrated Circuits
IC	Integrated Circuit
LANCE	Local Area Network Controller for Ethernet
LSI	Large Scale Integration
MARS	MAintainable Real-time System
ML	Message Loss
MM	Message Mismatch
MMU	Memory Management Unit
MSI	Medium Scale Integration
NMI	Non-Maskable Interrupt
NOAM	NO Application level Mechanisms
NOAMN	NO Application level Mechanisms, No NMIs

OS	Operating System
PF	Pin-Forcing
SEMC	Single Execution Message Checksum
SEMCN	Single Execution Message Checksum, No NMIs
SEU	Single Event Upset
SF	System Failure
SRAM	Static Random Access Memory
SRU	Smallest Replaceable Unit
SSI	Small Scale Integration
SWIFI	SoftWare Implemented Fault Injection
TDMA	Time Division Multiple Access
TEMC	Triple Execution Message Checksum
TSC	Time Slice Controller
UART	Universal Asynchronous Receiver Transmitter
UEE	UnExpected Exception
VLSI	Very Large Scale Integration
WS	Warm Start