Comparison and Integration of Three Diverse Physical Fault Injection Techniques

Johan Karlsson
Peter Folkesson
Chalmers University of Technology

Jean Arlat
Yves Crouzet
LAAS-CNRS

Günther Leber
Johannes Reisinger
University of Technology Vienna

This report describes and compares three physical fault injection techniques — pin-level fault injection, heavy-ion radiation, and electromagnetic interference (EMI) — and their use in the validation of MARS, a fault-tolerant distributed real-time system. The objectives of this study are: (i) to validate the fault-tolerance features of the MARS system, and (ii) to gain a better understanding of the features and impact of the three fault injection techniques. Coverage measures and validation goals are defined and discussed. In addition, the target system and the specific experimental set-ups for the three fault injection techniques are described.

1 Introduction

The dependability assessment of a fault-tolerant system is a complex task that requires the use of different levels of evaluation and related tools. Besides and in complement to other possible approaches such as proving or analytical modelling whose applicability and accuracy are significantly restricted in the case of complex fault-tolerant systems, *fault-injection* has been recognized to be particularly attractive and valuable. Indeed, by speeding up the occurrence of errors and failures, fault injection is in fact a method for *testing* the fault tolerance algorithms/ mechanisms with respect to their own specific inputs: *the faults*.

Fault injection has been used to test both hardware and software, but most studies have concerned the validation of fault-tolerant systems aimed at tolerating physical faults [Arlat 1992]. Fault injection can be applied either on a simulation model of the target fault-tolerant system (e.g., [Goswami and Iyer 1992; Jenn et al. 1994]) or on an actual hardware-and-software implementation (e.g., [Arlat et al. 1990b; Walter 1990])

Clearly simulation-based fault injection is desirable as it can provide early checks in the design process of fault tolerance algorithms/mechanisms. Nevertheless, it is worth noting that fault injection on a prototype featuring the actual interactions between the hardware and software dimensions of the fault tolerance algorithms/mechanisms supplies a more realistic and

necessary complement to validate their implementation in a fault-tolerant system. Until recently, most studies related to the application of fault injection on a prototype of a fault-tolerant system relied on physical fault injection, i.e., the introduction of faults through the hardware layer of the target system [Schuette et al. 1986; Gunneflo et al. 1989; Arlat et al. 1990a]. A trend favouring the injection of errors through the software layer for simulating physical faults (i.e., software-implemented fault injection) is currently emerging (e.g., see [Segall et al. 1988; Kanawati et al. 1992]). Although such an approach facilitates the application of fault injection, the correspondence between the types of errors that can be injected this way, and the actual faults is not yet confidently established. In spite of the inherent difficulties in developing adequate support environments and realizing experiments, physical fault injection enables real faults to be injected in a very close representation of the target system without any alteration to the software being executed.

Among the large number of experiments reported concerning physical fault injection, all used widely different techniques and/or were applied to distinct target systems. This significantly hampers the possibility to identify the difficulties/benefits associated to each fault injection technique and to analyse the results obtained.

The study being reported here describes a unique joint effort involving Chalmers University of Technology, LAAS-CNRS and the University of Technology Vienna that has been launched within the framework of the PDCS-2 project of the ESPRIT programme. This study relies on two major objectives.

The first one is to get a better understanding of the impact and features of the three physical fault injection techniques that are considered and in which the sites have gained expertise in developing and applying dedicated experimental tools or in using standard support environments, respectively: heavy-ion radiation, pin-level injection and electromagnetic interferences (EMI).

The distributed fault-tolerant system architecture MARS developed by UT-Vienna is being used as the target system to carry out these experiments. Thus, the other driving objective is to evaluate the coverage of the built-in fault-tolerance features of the MARS system. A distributed testbed architecture featuring five MARS nodes and a common test scenario have been implemented at all three sites to perform a coherent set of experiments.

The remaining part of this paper is decomposed into eight sections. Section 2 briefly describes the main fault tolerance features of the MARS architecture and the structure of the MARS nodes. The experimental assessment of error detection and fault tolerance coverage is discussed in Section 3. Section 4 depicts the common testbed set-up being used by all sites to carry out the experiments. Section 5 specifies the predicates characterizing the behaviour of the target system in the presence of injected faults and defines the estimators being considered to evaluate the coverage provided by the error detection and recovery mechanisms. Section 6 presents the main features of the fault injection techniques being applied, heavy-ion radiation, pin-level injection and EMI, respectively. A brief comparison of the fault injection techniques is presented in Section 7. Section 8 identifies the expected outputs and insights that can be derived from the joint application of these various techniques on the same experimental set-up. Finally, some concluding remarks are provided in Section 9.

2 The MARS Architecture

A fault tolerant distributed real-time system can be built up with a number of autonomous, fail-silent [Powell et al. 1988] processing nodes that are interconnected by a real-time network and communicate by the exchange of messages. The MARS system is the realization of such an approach.

In MARS all active and passive components are replicated in order to prevent a single failure of such a component from causing a system failure. Up to three processing nodes execute identical software and form a Fault-Tolerant Unit (FTU). The FTUs communicate via the real-time network that is implemented as a replicated broadcast channel (see Figure 1).

In the next subsections fault tolerance and timeliness issues at system-level are discussed first, then the structure of a MARS node aimed at supporting these features is described.

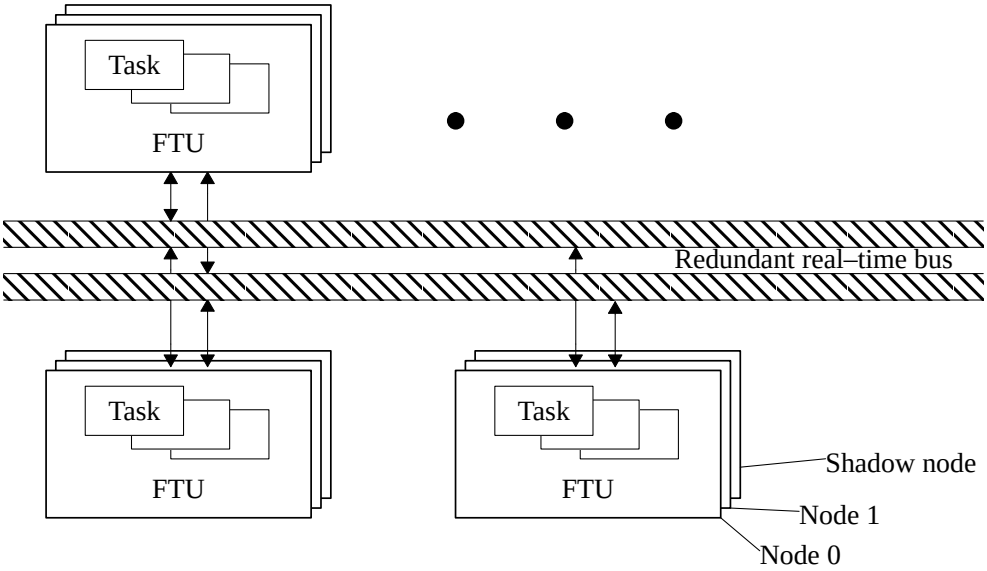

Figure 1: System structure of MARS

2.1 Fault Tolerance

MARS uses a two-layered mechanism to achieve fault-tolerance. The bottom layer (node layer) is responsible for error detection and error confinement (i.e., node shutdown on error). These functions are carried out by the processing nodes, which are therefore called ‘fail-silent’. The task of fail-silent nodes is to detect all internal errors and to prevent their propagation. Therefore, the top layer (system layer) needs not to care about erroneous data, it only has to provide enough redundancy to tolerate (silent) failures of parts of the system. The major functions of the top layer are handling of redundant data and reconfiguration of the system in case of a node failure [Grünsteidl and Kopetz 1991].

To achieve a deterministic timing behaviour even in the presence of faults, the MARS system uses active redundancy for all processing and communication activities: each process is executed simultaneously at all nodes of an FTU and each message is transmitted quasi-simultaneously on each of the broadcast channels. Due to the fail-silence property the results

of all three nodes of an FTU are assumed to be correct and may be used interchangeably. Since we need only two nodes to tolerate a single failure of a fail-silent node (i.e., the loss of a message), the optional third node, the shadow node, does not transmit any message on the real-time network as long as both active nodes are operational. Only if an active node fails, the shadow node immediately starts to transmit its results, thus restoring the initial degree of redundancy.

2.2 Timeliness

MARS is strictly time triggered, in other words, each relevant action of the system is scheduled before operation. This property allows to guarantee the proper timing behaviour of the system already during the design phase of an application. The following must be considered:

- the points in time when a node is allowed to send a message (following the time-triggered paradigm, MARS uses a TDMA[1] protocol for communication) and the types of messages which 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 reintegration of failed nodes; although these actions are rarely performed, they have to be incorporated into the schedule to prevent a properly detected fault from affecting the correct timing behaviour of the system.

It is not necessary to consider error handling at single-node level in the timing analysis because the detection of an error causes a reset of the processing node. No dedicated recovery actions are performed within a node.

The existence of a precise global time base is a basic requirement for a time triggered system, because the actions within different processing nodes of the system must be synchronized. Global time is maintained by a distributed, fault tolerant clock synchronization algorithm.

The static structure of MARS forces the system designer to investigate the timing behaviour of all parts of the system carefully; we have to know time bounds for all processing and communication activities in order to assign time slots to them. In particular, the following timing parameters must be known:

- the maximum execution time of each process, considering the architecture and speed of the processor, pipelining, caching, and memory wait states [Puschner and Koza 1989],
- the maximum time for communication,
- the operating system overhead, and
- the overhead of hardware activities influencing the timing behaviour of the node.

[1] Time Division Multiple Access

2.3 The Processing Node

In order to support the MARS-concepts of real-time and fault-tolerance in an optimal way, a special-purpose processing node was designed [Reisinger and Steininger 1994; Steininger and Reisinger 1991]. In the following the structure of the processing node and the error detection mechanisms will be explained.

2.3.1 Structure of the Node

The single board implemented node consists of two independent processing units (see Figure 2), the *application unit* and the *communication unit*. Each unit is based on a 68070 CPU, a processor resembling the 68000 and including a memory management unit, a two-channel DMA controller, a UART (universal asynchronous receiver and transmitter) interface (RS232), an Inter-IC (I^2C) bus, and an interrupt controller [Philips Semiconductors 1991]. Further every unit features a 16 bit parallel I/O-port and an EPROM. The application unit also contains a dynamic RAM, and two bidirectional FIFOs, one of which serves as an interface to external add-on hardware, the other one connects the application unit to the communication unit. Additional hardware for the communication unit comprises also a Static Random Access Memory (SRAM), two Ethernet controllers, two Clock Synchronization Units (CSU) required for maintaining a global time base, and a watchdog timer.

2.3.2 Error Detection Mechanisms

Three levels of error detection mechanisms (EDMs) are distinguished: (i) the hardware EDMs, (ii) the system software EDMs implemented in the operating system [Reisinger 1993; Reisinger 1992; Kopetz et al. 1992] and support software (i.e., the Modula/R compiler [Vrchoticky 1992]), and (iii) at the highest level end-to-end EDMs. These mechanisms are described in the following subsections.

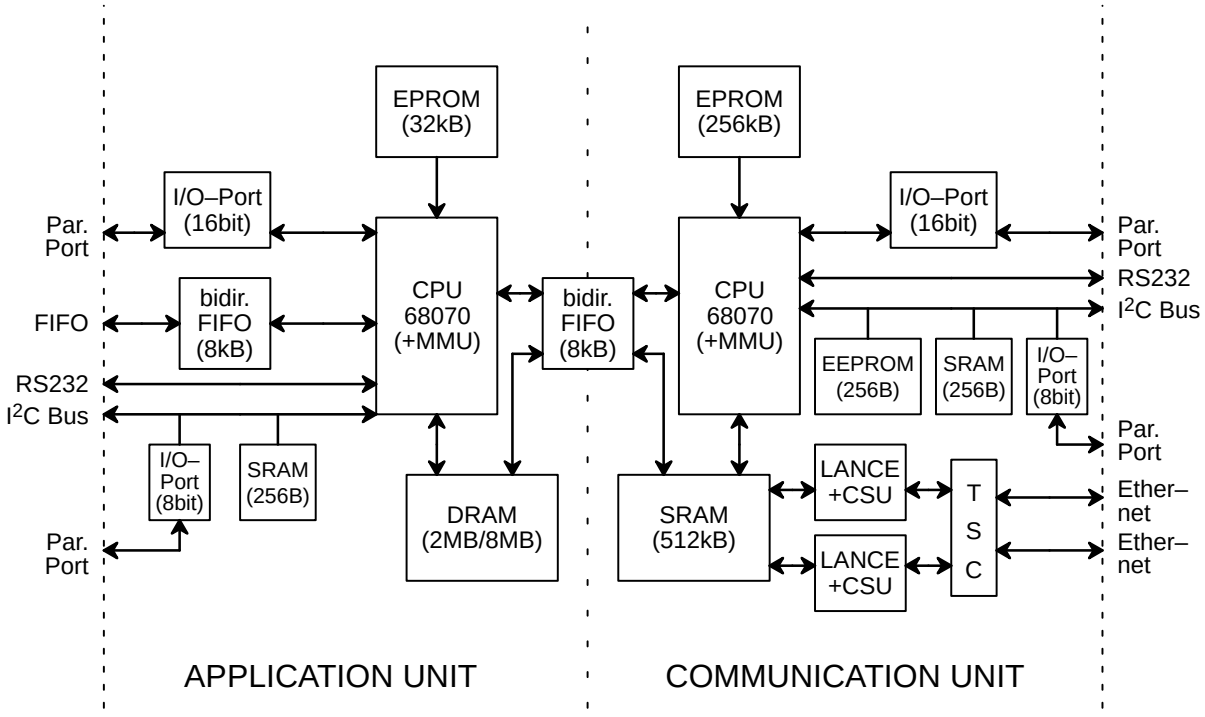

Figure 2: Block diagram of the processing node

2.3.2.1 Hardware EDMs

Whenever an error is detected by one of the Hardware EDMs, in general, a Non-Maskable Interrupt (NMI) is raised and the CPU will then wait for a reset issued by a watchdog timer. This watchdog timer is the only device, which may cause a reset of all devices including the CPU.

Two categories of hardware EDMs can be distinguished: the mechanisms provided by the CPU and those provided by special hardware on the processing board. The CPU built-in EDMs [Philips Semiconductors 1991] are:

Bus error: This is detected by the on-chip Memory Management Unit (MMU) of the processor and occurs when access to the specified address is not allowed.

Address error: The processor can detect misaligned access to memory, e.g., a long word is accessed at an odd address.

Illegal opcode: An instruction is read, which is not a valid instruction for the processor.

Privilege violation: is an attempt to execute a privileged instruction in user mode.

Division by zero: indicates, that a divide operation was performed with divisor zero.

These errors cause the processor to jump to the appropriate exception handling routines, which save the error state to a non volatile memory and then restart the node. On restart a detailed error description is written to a serial port.

The following error detection mechanisms are implemented by special hardware on the node:

Silent shutdown of the CPU of the communication unit is detected by a watchdog timer, which resets the node when the CPU does not change an input of this timer within 1.6 milliseconds.

Power failure is detected by an IC, which monitors the DC voltage delivered to the node. If a voltage drop occurs, an NMI is raised.

Parity error is checked on memory (DRAM) access or when data is transferred to/from a FIFO memory. This is achieved by add-on hardware — the parity checkers — which raise an NMI whenever such an error is detected.

FIFO over/underflow is signalled when either the FIFO (first-in first-out) memory becomes flooded or data is read from an empty FIFO.

Access to physically non-existing memory is caught by the address decoder logic, which raises an NMI in case of such an access.

Write access to the real-time network at an illegal point in time: For detection of this error a special processor — the time slice controller — is attached to the board, which raises an NMI when transmitting to one of the network-channels is attempted at a point in time, when the node is not allowed to transmit data.

Error of an external device is an error input, which can be used by trusted external devices to

raise error conditions. Activating this input means also that an NMI is generated.

Error of other unit is the NMI input line of the other processor on the node.

An NMI condition leads to the same exception handling as the built-in CPU error detection mechanisms and can only be cleared by resetting the node's processor, which is done by means of the watchdog timer.

2.3.2.2 System Software Error Detection

The EDMs implemented by system software include:

- value range overflow of a variable,
- loop iteration bound overflow,
- processing time overflow,
- various checks on data, done by the operating system, and
- various assertions coded into the operating system.

The first two of these mechanisms are built into the compiler, which produces application code, the Compiler Generated Run-Time Assertions (CGRTA), that can detect those errors; the others are built into the operating system as assertions or as integrity checks on data. When an error is detected by one of these mechanism, a trap instruction is executed, which then leads to a node restart.

2.3.2.3 End-to-end Error Detection

The end-to-end EDMs include end-to-end checksums for message data and double execution of tasks. The end-to-end checksums are used to detect mutilation of message data and thus this is used for implementing the *extended fail-silence property* of the nodes, i.e., the node is also considered to be fail-silent when a detectably corrupt message is sent, because the receiver then will discard the message. Execution of tasks in time redundancy can detect errors caused by transient faults that cause different output data of the two instances of the task. Combined with the concept of message checksums, task execution in time redundancy forms the highest level in the hierarchy of the error detection mechanisms. These errors also trigger the execution of a trap instruction, which causes a reset of the node.

3 Coverage Assessment

For characterizing the fault/error processing action of a fault tolerance mechanism (detection, retry, diagnosis, localization, reconfiguration) one can define a predicate P such that P = 1 if the action is appropriate and P = 0 otherwise. Figure 3 gives an example of such a predicate concerning an error detection mechanism.

The interval [0,*T*] determines the observation domain: the initial time corresponds to the time

of fault injection and it is necessarily censored on the right in order to maintain a finite maximal observation time characterizing the end of the experiment.

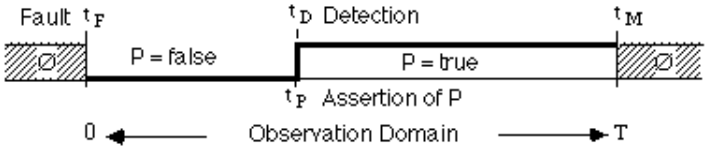

Figure 3: Example of predicate

Coverage is then defined as the *cumulative distribution* of the time interval between the occurrence of a fault and its correct handling by a fault tolerance mechanism [Dugan and Trivedi 1989; Arlat et al. 1990a] (i.e., the assessment of P). Figure 4 depicts a typical example of the empirical distribution of the coverage.

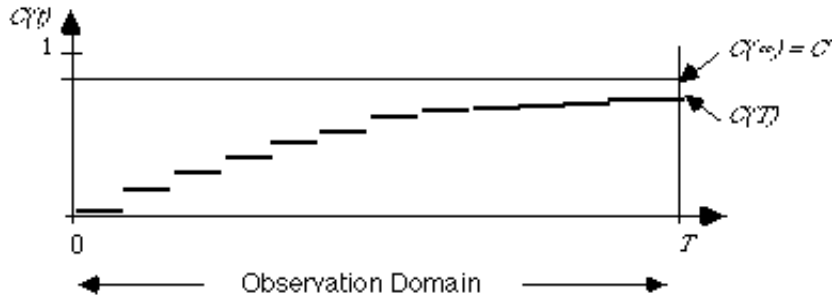

Figure 4: Empirical coverage distribution

The determination of the interval [0,*T*] is essential for the estimation of coverage: the measurement of the time of occurrence of the monitored event (e.g., the detection of an error) is conditioned by this choice. In particular, it can be seen that the asymptotic value of $C(t)$ (i.e., the coverage factor *C*) is pessimistically estimated by $C(T)$. This also leads to an uncertainty with respect to the estimation of timing characteristics, e.g., the time interval between the injection of a fault (or of its activation as an error) and its detection (assertion of P). In practice, preliminary experiments are to be carried out to help select *T.* Although timing measurements are of definite interest to fully characterize the fault tolerance mechanisms (see e.g., [Arlat et al. 1993] for a detailed analysis of the impact of error detection latency on dependability measures), such a measure cannot be easily obtained with all of the injection techniques used in the study, we focus on the *asymptotic* value of this distribution, which is usually called the *coverage factor.*

Lack of coverage may be due to two major types of deficiencies:

- faults affecting the design/implementation of the fault tolerance mechanisms with respect to the hypotheses considered,

- consideration of fault hypotheses different from the ones most probably encountered in the operational phase.

Fault injection is only suitable to evaluate the first type of deficiency. Other forms of experiments and measurements are necessary to cope with the second one.

The estimation of the coverage factor is based on the observation of the assertion of predicates during a series of fault injection experiments. A predicate may be defined for each of the detection mechanisms tested or these can be combined into a single global predicate.

In general, the input space of a fault-tolerant system is composed of: (a) the *functional* inputs to the system (or *activation set A*), and (b) the non-functional inputs composed of the *fault set F* to which the system should be tolerant. In the experiments carried out in this study we essentially emphasize sampling in the *F* set; the *A* set corresponds to a fixed representative application program.

In practice, it is expected that the injection techniques considered will correspond to distinct and complementary samples of the *F* set. Furthermore, it is important to note that the way the samples are selected is very much different for the three techniques. As shown in [Powell et al. 1993] the sampling technique and/or the expression of the estimator to be used has a very big impact on the accuracy and precision of the evaluation of the coverage. However, as it is not possible to precisely control or monitor the sampling of the *F* sets for two of the fault injection techniques used (heavy-ion and EMI), we will simply rely on evaluations obtained by considering a simple estimator characterizing *coverage proportion*, defined as the ratio of the number of observed assertions over the total number of experiments. Also a large number of experiments are needed in order to increase the confidence of the estimation.

The measures considered for the evaluation of the error detection mechanisms incorporated in the MARS architecture are defined in Section 5.

4 Common Experimental Set-up

The experimental set-up used by all sites consists of five MARS nodes and is similar to the one used in [Damm 1988]. The workload is a realistic control application.

4.1 Hardware Configuration

For these experiments five MARS-nodes are needed (see Figure 5). One serves as a gateway between the department's local area network and the MARS-bus and is required for loading the entire application and for reloading failed nodes. Another node is used for simulating input data for a task, which is used for testing. This test task is performed on two actively redundant working nodes, one of which serves as *golden node*, the other one is subjected to fault injection. The fifth node is used for checking the output of the two nodes performing the test task, for giving status data about the experiments, and for controlling the power supply of the node under test and the fault injector.

Further a host computer is used, which is controlling the experiments, i.e., reloading failed nodes and collecting data from the experiment for further analysis.

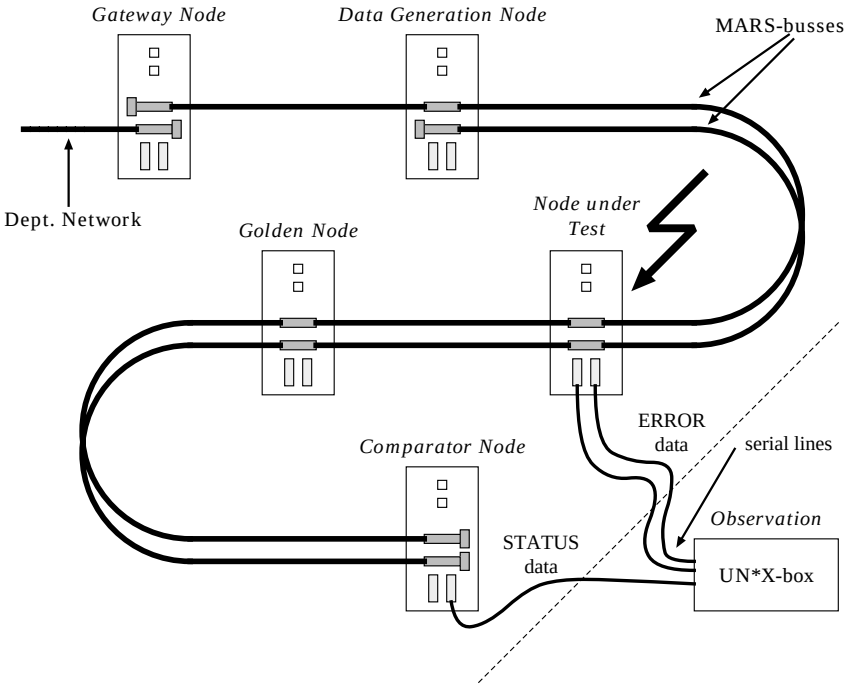

Figure 5: Hardware configuration

4.2 Test Application

A typical real-time application was chosen for the fault injection experiments — a control-problem, in order to have a realistic workload, since error-detection coverage factors are highly application dependent. The control task was taken from the rolling ball experiment [Kopetz et al. 1991] in which a ball is kept rolling along a circular path on a tiltable plane by controlling the two horizontal axes of the plane by servo motors and observing the position of the ball with a video camera. The tiltable plane and the camera are not present in our set-up; instead, the data from the camera is simulated by a data generation task. An additional task was provided, which compares the results of the two actively redundant computing nodes, both of which execute the control task.

As already mentioned, the application used for the fault injection experiments basically consists of these three tasks (see also Figure 6):

1. The *data generation task* generates the input data for the control task. The input data include the nominal and actual values of the position, speed and acceleration of the ball.

2. The *control task* , which does not preserve any data ot state information between its periodic executions, receives this emulated data from the data generation task and performs calculations on these data, i.e., calculates the desired acceleration for the ball.

3. The *comparator task* receives the results delivered by two nodes, which run the control-task in active redundancy, and compares them. This task gives also status information about the experiment, and assists in controlling a fault injection device (i.e., it indicates when fault injection may take place) and the power supply of the node under test.

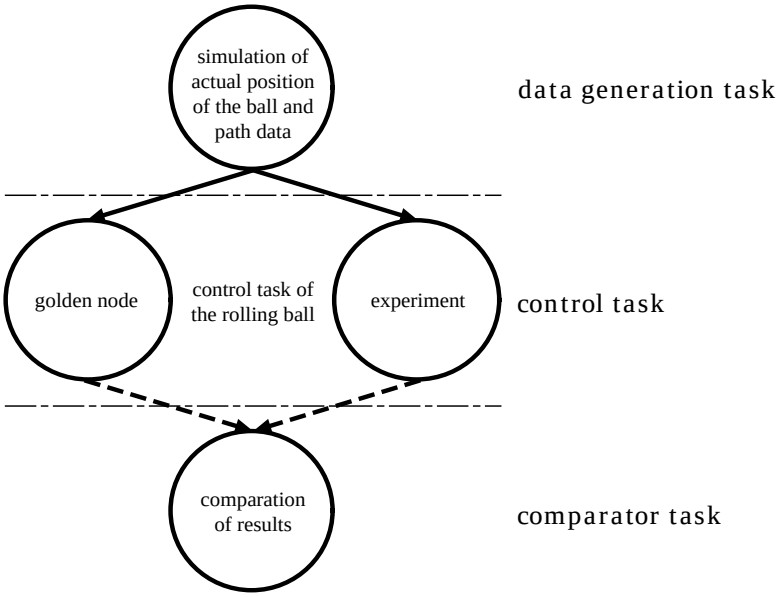

Figure 6: Tasks and message flow

The entire application has a period of 40 ms, i.e., all application tasks are started every 40 ms and hence produce a result in the same time interval. The application software is written in Modula/R [Vrchoticky 1992] a Modula/2 like programming language with real time support for MARS.

5 Measurements

In order to have a common measure for the evaluation of the fail-silence property, the following estimator will be used:

$$\text{FailSilenceCoverage} = \frac{\text{Number of Detected Errors}}{\text{Number of Occurring Errors}}$$

where *detected* errors are those, which are detected by the EDMs in the node subjected to fault-injection itself or by means of message checksums at the receiving node, and *occurring* errors are those, which are observed by the comparator node.

5.1 Predicates

An experiment consists of several experiment runs. In the beginning of an experiment all MARS-nodes are loaded with the application described in Section 4.2. When all the nodes are operating fault-injection takes place until the node fails in one of the ways described later. Then the node under test is shut down by the comparator node by removing its power, in order to clear all error conditions for the new experiment run. After some time power is reinstalled and the node under test is reloaded for the next experiment run.

The node under test can basically fail in three ways. First, the node's EDMs may detect an error. In this case the node stores the error condition into non-volatile memory and resets itself by means of the watchdog timer. In the second case, the comparator node detects a fail-silence violation, i.e., a mismatch between the messages sent by the node under test and the golden node. The third way for the node to fail is not to deliver the expected application message(s) for one or several application cycles.

If any other node than the node under test fails, the entire system is halted, in order to allow a detailed analysis as to why the system failed.

On every restart the node under test writes its previously saved error information, if available (i.e., if an error was detected by the node's EDMs), and information about its state to the serial ports, where it can be read and stored for further processing. From this data the following events can be derived:

Coldstart (CS): Coldstart (power on) of the node under test is made after every experiment run, except when a system failure occurred.

Internally induced warmstart (IWS): Warmstart (reset) of the node under test caused by detection of an error by the node's EDMs.

Externally indicated warmstart (EWS): Warmstart (reset) of the node under test because of detection of an incoming or outgoing link failure by means of the top layer of the fault-tolerance mechanism (i.e., the membership protocol [Kopetz et al. 1989]).

Message loss of node under test (C1): One message from the node under test was lost (i.e., not received by the comparator node).

Fail-silence violation (C2): Reception of differing messages from golden node and node under test.

System Failure (F): Failure of golden node, data generation node, or comparator node.

The above described events results in the failure scenarios as shown in Figure 7. Each of these failure scenarios is classified into one of the following failure types: IWS/EWS, C1, C2 or F .

(a) If a node other than the node under test fails, the entire system is halted in order to make it possible to diagnose this unexpected behaviour. At this point the entire experiment is stopped. This is an F failure.

(b) This is the normal case. An internally triggered or externally indicated warmstart occurs and as a result the node immediately stops sending messages. At the end of the experiment run, the comparator node initiate a coldstart of the node under test. This is an IWS/EWS failure.

(c) If the comparator node does not receive one or more messages from the node under test, it will be shut down. This behaviour is not a fail-silence violation, because no erroneous data is sent, but it also cannot be regarded as normal operation. This is a C1 failure.

(d) This is similar to case (c) except that an internally triggered or externally indicated warmstart occurred before the coldstart. This is an IWS/EWS failure.

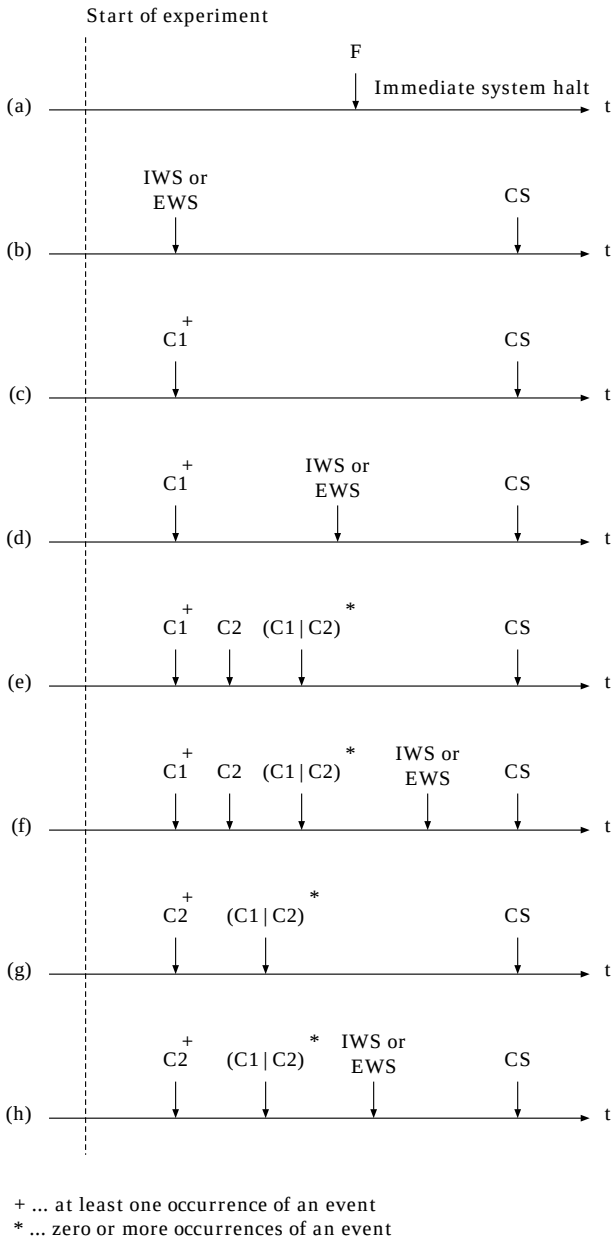

Figure 7: Failure scenarios

(e) After a loss of one or more messages, a fail-silence violation occurs. This is a C2 failure.

(f) This case is similar to (e) except that the EDMs detected an error too late. This is a C2 failure.

(g) One or more C2 events occur, followed by zero or more C1 and/or C2 events. This is a C2 failure.

(h) This is the similar to (g) except that the node detects an error too late. This is also a C2 failure.

Given the above failure types the definition for Fail Silence Coverage can be expressed as:

$$\text{FailSilenceCoverage} = 1 - \frac{N_{C2} + N_F}{N_{tot}}$$

where N_{C2} and N_F is the number of failures of type *C2* and *F*, respectively, and N_{tot} is the total number of experiment runs. The failures of type F will be accounted for only if they are caused by a lack of fault isolation between the affected node and node under test.

5.2 Experiments

A first set of experiments will be carried out to obtain a fail silence coverage, only covering the hardware and system software error detection mechanisms, to see how effective these EDMs are. To be able to test and derive an effectiveness for the end-to-end EDMs, experiments will be carried out with all EDMs enabled, and subsequently with hardware and system software EDMs, and message checksums enabled, i.e., double execution of tasks disabled. Finally experiments will be conducted for obtaining the effectiveness of the hardware EDMs, which are implemented by special hardware and raise an NMI (non maskable interrupt), whenever an error is detected, i.e., experiments with different hardware EDMs disabled will be carried out.

6 Fault Injection Techniques

6.1 Heavy-Ion Radiation

Heavy-ion radiation from a Californium-252 can be used to inject Single Event Upsets (SEUs), i.e., bit flips at internal locations in integrated circuits. The heavy-ion method has been used to evaluate several hardware- and software-implemented error detection mechanisms for the MC6809E microprocessor [Gunneflo et al. 1989; Miremadi et al. 1992]. A comprehensive description of the heavy-ion fault injection technique is given in [Karlsson et al. 1994].

A major feature of the heavy-ion fault injection technique is that faults can be injected into VLSI circuits at locations which are impossible to reach by other techniques such as pin-level and software-implemented fault injection. The faults are also reasonably well spread within a circuit, as there are many sensitive memory elements in most VLSI circuits. Thereby, the injected faults generate a variety of error patterns which allows a thorough testing of fault handling mechanisms.

The fault mechanism can shortly be described as follows. When a heavy-ion penetrates a semiconductor material, it creates a track of electron-hole pairs. If the particle passes through a reverse biased pn-junction, the high electric field present there will cause the deposited charge to induce a transient voltage pulse in the associated circuits. If the particle hits a sensitive region inside a memory element (flip-flop or latch), the voltage transient will immediately change the state of that memory element.

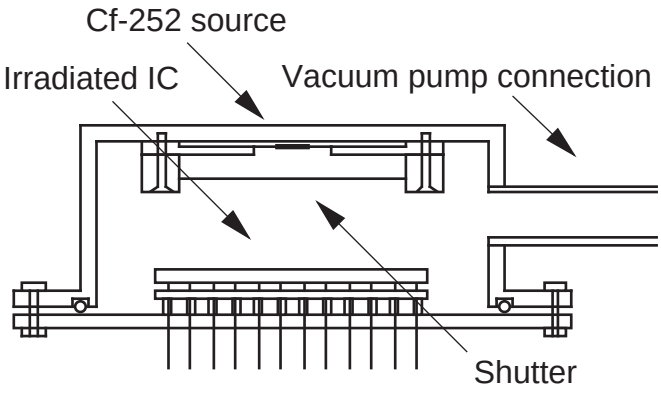

Figure 8: Cross-sectional view of the miniature vacuum chamber

Heavy-ions may also induce transients in combinational logic, but the probability that such a transient becomes latched into a memory element is low. It is reasonable to assume that for most circuits no more than a few percent of the bit flips are caused by transients in combinational logic [Lidén et al. 1994]. Consequently, a very large majority of the bit flips are caused by direct hits in memory elements. In most cases, a single heavy ion only affects a single bit, although there are circuits for which single ions may cause bit errors in two physically adjacent memory elements[1] [Johansson 1993].

As the heavy ions are attenuated by air molecules and other materials, irradiation of circuits must be performed in a vacuum. Any packaging material that covers the chip must also be removed. To facilitate the irradiation of circuits in a system, a miniature vacuum chamber has been developed, which contains the Cf-252 source and the irradiated circuit, see Figure 8. The circuit'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.

The distance between the irradiated circuit and the Cf-252 source is approximately 35 mm (the distance varies slightly depending on the package type). By inserting extension tubes between the bottom plate and the rest of the miniature vacuum chamber, it is possible to increase the distance and thereby reduce the flux of heavy-ions impinging onto the chip. The vacuum chamber also contains a electrically manoeuvred shutter, which is used to shut radiation on and off. A commercially available Cf-252 source with a nominal activity of 37 kBq (1 kBq = 1000 disintegrations per second) is used. About three percent of the disintegrations generate fission fragments (i.e. heavy-ions); the rest generate alpha particles. Most circuits are not affected by the alpha particles as they deposit much less charge than the fission fragments.

Besides SEUs, heavy-ion radiation may also cause latch-ups in CMOS circuits. A latch-up is the triggering of a parasitic four layer switch (npnp or pnpn) acting as a silicon controlled rectifier (SCR), which may destroy the circuit due to excessive heat dissipation. A latch-up is indicated by an drastic increase in the current drawn by the circuit. To protect circuits from latch-up, we have developed a device which monitors the current drawn by the irradiated

[1] Such memory elements may not necessarily belong to the same register.

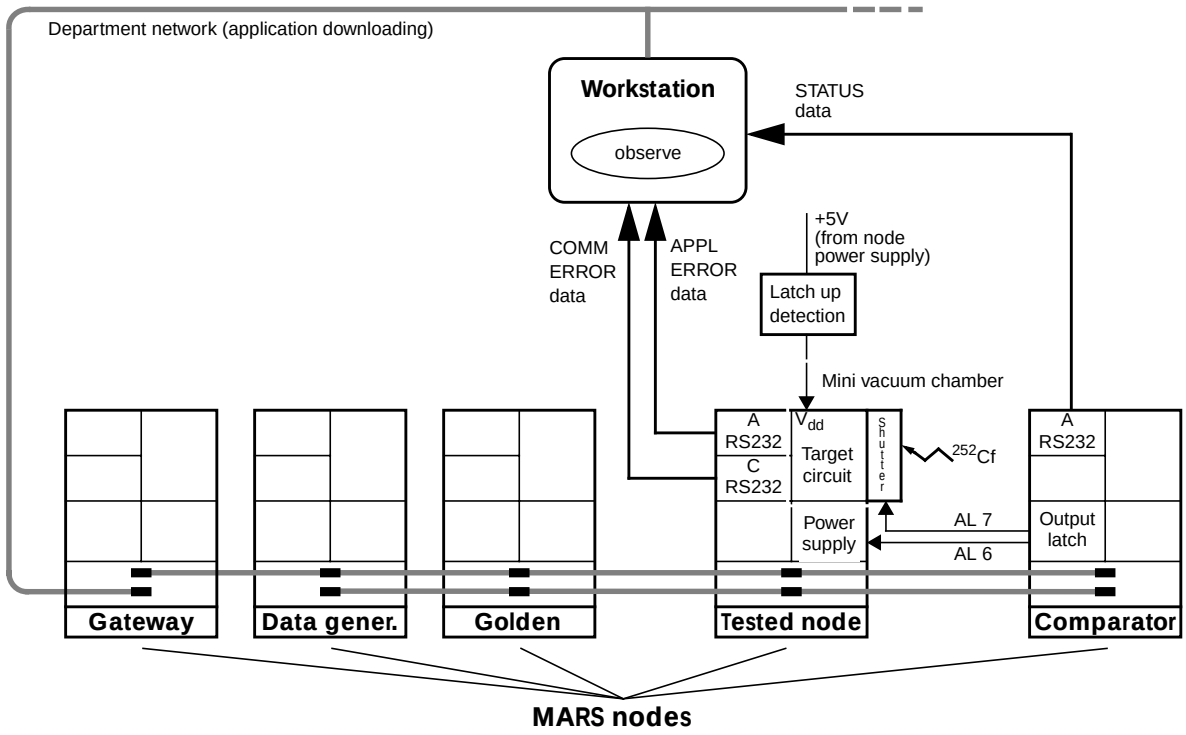

Figure 9: Basic set-up for the heavy-ion experiments

circuit and immediately turns off the power if the current exceeds a threshold value.

In the validation of the MARS system three key components will be irradiated: the CPU of the application unit, the CPU of the communication unit and the LAN controller circuit for the MARS bus. Both the application and the communication unit use the 68070 CPU manufactured by Philips. The LAN controller is Am7990 manufactured by Advanced Micro Devices.

We plan to implement two experimental set-ups for the heavy-ion experiments; a basic set-up and an advanced set-up. The basic set-up only allows assessment of asymptotic coverage, while the advanced set-up also makes it possible to measure the error detection latency.

A block diagram of the basic set-up is shown in Figure 9. The experiments are controlled by the comparator MARS node and a UNIX workstation. The workstation is also responsible for data collection. When the comparator node detects an error (error type C1 or C2, see Section 5.1), it reports the error type to the workstation and turns off the power to the node under test with the signal AL 6. It also closes the shutter mechanism in the miniature vacuum chamber with the signal AL 7 to stop the irradiation of the circuit. It then turns on the power again to restart the node under test. Upon restart the application unit and the communication unit in the node under test send error data to the workstation via two serial lines. (If the error is not detected by the node under test itself, then the node has no error information available and sends only a status message.) Once the node under test has been restarted the workstation immediately starts to download the application. When the application has been restarted, the comparator node opens the shutter and a new experiment begins.

A block diagram of the advanced set-up is shown in Figure 10. This set-up uses a comparator

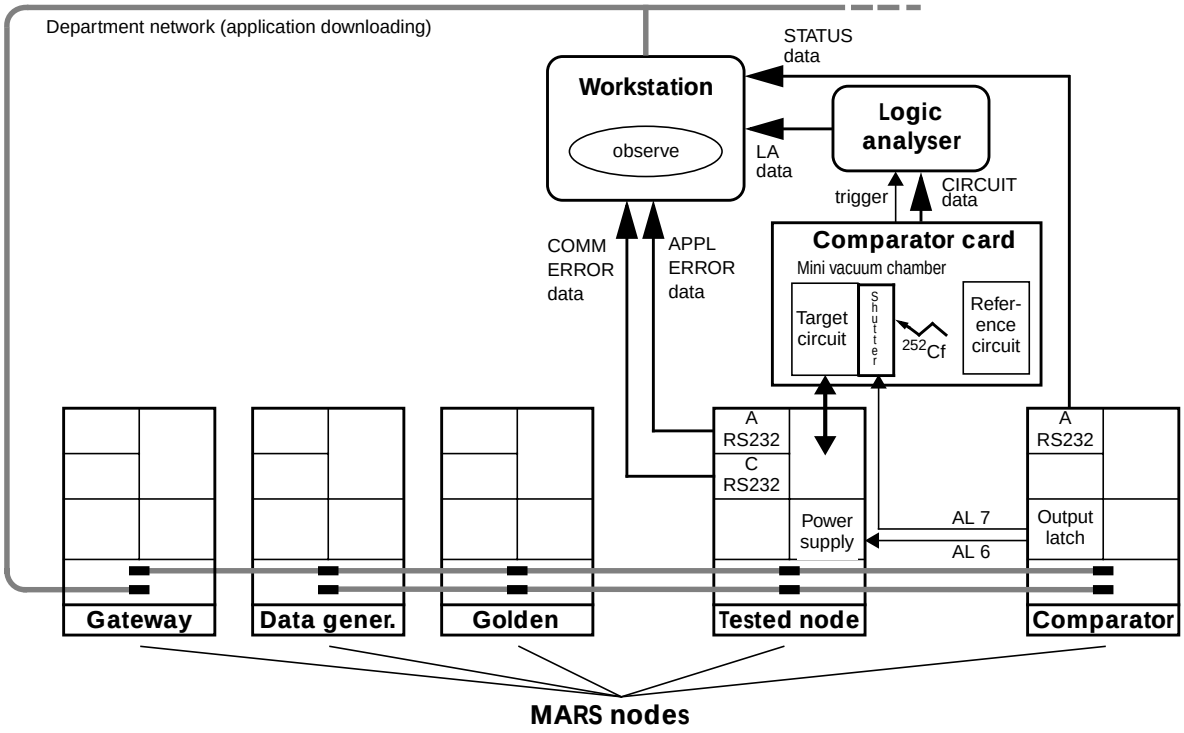

Figure 10: Advanced set-up for the heavy-ion experiments

card for comparing the outputs of the irradiated circuit with the outputs of a reference circuit not subjected to fault injection. The two ICs are fully synchronized and their outputs are compared for each clock cycle. In this way errors occurring on the outputs of the irradiated circuit are immediately recognized. The error detection latency is determined by means of a logic analyser that measures the time between the occurrence of an error and the detection of that error. The logic analyser can also be used for detailed study of the error behaviour of the target IC.

6.2 Pin Level Injection

Pin-level fault injection, i.e., the injection of faults directly on the pins of the IC of a prototype was until now the most widely applied physical fault injection technique. It has been used for (i) the evaluation of the coverage of specific mechanisms (in particular for error detection by means of signature analysis [Schuette et al. 1986], and (ii) the validation of fault-tolerant distributed systems (e.g., [Damm 1986; Walter 1990]). Flexible tools supporting some general features have been developed (e.g., MESSALINE at LAAS-CNRS [Arlat et al. 1990a] or RIFLE [Madeira et al. 1991] at the University of Coimbra).

These tools support one (or both) of the possible implementations for pin-level fault injection:

- *forcing*, where the fault is directly applied by means of multi-pin probes on IC(s) pin(s) and associated equipotential line(s),
- *insertion*, where the IC(s) under test is(are) removed from the target system and inserted

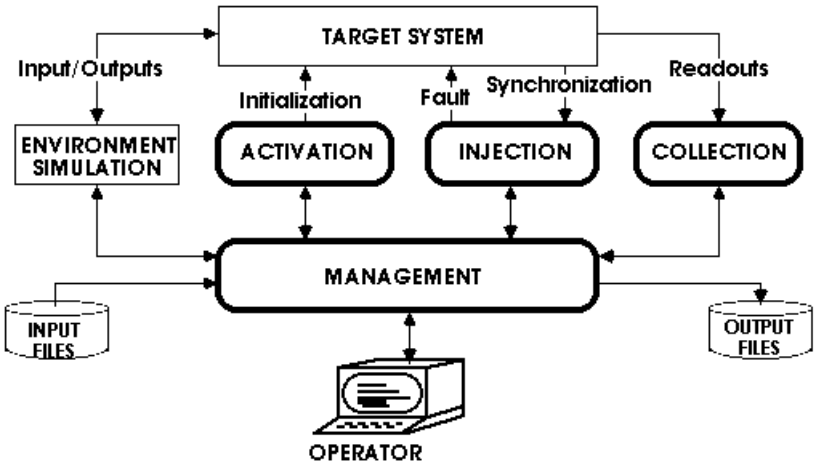

Figure 11: General architecture of MESSALINE

on a specific box where transistor switches ensure the proper isolation of the IC(s) under test from the system.

The most current fault models are stuck-at (0 or 1). Concerning temporal characteristics of the fault models, if it was admissible, several years ago, to use primarily permanent faults in the case of a system built with low scale of integration ICs, it is no longer possible with today's technology; indeed, this fault model is no longer representative of consequences of internal faults affecting LSI ICs. It is mandatory to be able to inject temporary faults on the pins of the ICs to simulate the consequences of such faults on the pins of the faulted IC(s).

Another point is to recognize the fact that the increase in the frequency rate of current VLSI ICs poses some problems of parasitic mutation at the level of the injection fixtures; to our perception, this seems to be the main reason for the current trend favouring the use of the forcing technique as this technique raises fewer problems of this type.

The tool MESSALINE that has been developed to facilitate the definition and realization of pin-level fault injection experiments will be used to test the MARS architecture. MESSALINE is a flexible tool capable of adapting easily to various target systems and to different measures [Arlat et al. 1990b]. Figure 11 presents the architecture of MESSALINE and the basic testbed configuration when connected to the target system (a physical hardware/software prototype). MESSALINE is made up of four modules: fault *injection*, target *activation*, readout *collection*, and *management* of the test sequence.

The injection module includes 32 injection elements supporting forcing and insertion techniques. In addition to stuck-at-0 and stuck-at-1 faults, the fault models supported include open connection and inverted signal in the case of insertion, and bridging in the case of forcing. It is worth noting that for these faults, the timing characteristics (duration and frequency of occurrence) are user-programmable and it is possible to control the injection elements by signals originating from the target system. Another important aspect to mention is that to each injection element is associated a device that enables the occurrence of an error (activation of a fault) to be identified at the point of injection (forcing: current sensing, insertion: comparison of signals on both sides of the switch). This capability is useful to ensure that the injected faults are actually activated and to perform a direct measurement of the fault

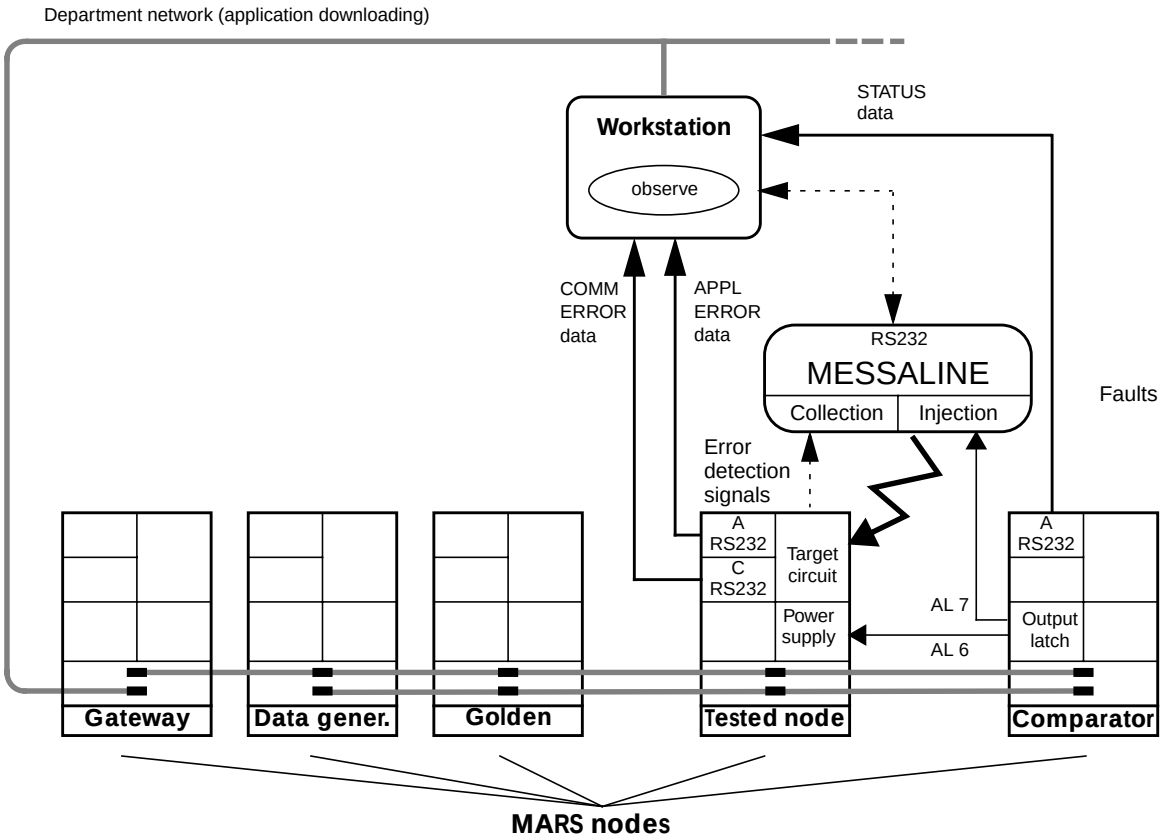

Figure 12: Set-up for the pin-level experiments

dormancy.

MESSALINE manages the test sequence by means of command files (board description, IC library, injection devices description, test sequence). From these files, depending on the parameters, a set of experiments can be derived deterministically or by a random assignation of the different parameters of a fault (multiplicity, fault types, timing characteristics, injected pins).

Figure 9: depicts the set-up for the pin-level fault injection experiments. As described for the heavy-ion experiments, the experiments are controlled by the comparator node and the UNIX workstation. The AL 6 signal is used to control the power of the experiment (fault-injected) node. The signal AL 7 is used to discontinue the injection of faults.

Additional measurements will include timing measurements (in particular fault dormancy as well as to some extent latency of specific EDMs— e.g., the NMI signal) this will be made be possible by means of the fault-activation recognition devices and the timers built into MESSALINE. This is shown by the dotted line entering the collection module (Figure 12). Another planned evolution of the set-up includes a serial link (shown by a dotted line also) connecting MESSALINE and the UNIX workstation; this will be used to enhance the controllability of each experiment, in particular with respect to (i) the sequencing of the successive steps within one experiment (selection of the fault parameters, reset of the node under test, downloading of the testbed or of the node under test, etc.), and (ii) the mapping between the results observed and the parameters characterizing the injected fault. Another

possible evolution will include the incorporation of a logic analyser to increase the observability of the error detection signals.

In the first set of experiments, the focus will be put on the test of the application microprocessor IC. More extensive fault injection experiments will then be carried out to test the other ICs of the node under test.

The forcing technique will be used to fault inject the node under test. In this case, it can be considered that all pins of the ICs connected, by means of an equipotential line, to a readily injected pin are tested as well. Accordingly, in the first set of experiments, to simplify the accessibility to the pins of the microprocessor, the target ICs will mainly be the buffer ICs directly connected to it.

The other characteristics of the injected faults are listed hereafter:

- one single IC will be fault injected at a time (the maximum number of pins faulted simultaneously — i.e., the *multiplicity* of the fault — will be limited to $mx = 3$[1]),
- uniform distribution over all combination of *mx* pins will be considered to select the *mx* faulted pins,
- only stuck-at-0 and -1 will be considered (all 0-1 combinations of *mx* pins will be considered equally probable),
- only transient faults will be injected in order to facilitate the comparison with the other techniques (the duration of the fault will be randomly distributed in the [1μs, 10μs] range).

A more sophisticated test strategy will be used for the subsequent experiments. As pointed out in [Powell et al. 1993]: (i) the net list describing the connection of the ICs and (ii) the relative failure rate of the ICs are useful information to respectively to assess the pin coverage achieved by the experiments conducted and to obtain an unbiased estimator of the coverage factor (see section 3). In the experiments conducted on the MARS system, both information will be used to establish a priori a representative sampling among the ICs of the node under test.

6.3 EMI

Since electro magnetic interferences are quite common in industrial plants featuring electrically powered machines and near spark ignition engines, we think that these are realistic faults and it was decided to use this kind of disturbances for fault injection. In these experiments a fault injector is used, which generates bursts conforming to the IEC 801-4 standard [Schaffner Instruments], i.e., the duration of the pulses is 15 ms, the period is 300 ms, and the frequency is 1.25, 2.5, or 5 kHz (see Figure 13). These bursts are similar to those, which arise when switching inductive loads with relays or mechanical circuit-breakers.

In a set of early experiments, the node under test was mounted between two plates which were

[1] The results obtained in [Gunneflo et al. 1989] substantiate such a choice; indeed, the experiments showed that more than 80% of first error occurrences on the pins of the radiated IC affected at most 3 pins. However, in the multiplicity distribution, emphasis will be put on single pin faults.

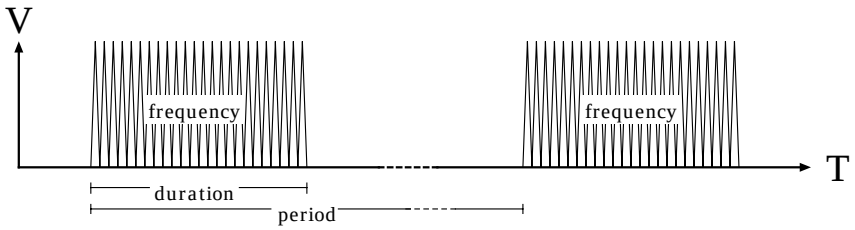

Figure 13: Electro-magnetic bursts

connected to the burst generator. There the real-time network turned out to be more sensitive to these disturbances than expected or even wanted. The wires connecting the board with the Ethernet-transceivers induced pulses, such that they caused the transceiver to indicate a jam-condition on the network. To solve this problem, the computer board had to be made more sensitive to the disturbances and the voltage level of the bursts had to be decreased. In order to achieve that, wires serving as antennas were attached to some of the node's ICs (the EPROM chip used for the lower order eight data bits, thus disturbing these data signals and the address signals on the board). In order not to have to use these wires, a special probe is now used instead for inserting the EMI-bursts into the system. The special probe is placed above a selected target circuit, e.g., the application unit's CPU as shown in Figure 14. In this way most of the injected faults will affect the target circuit. Note, however, that faults may also be injected into other circuits located near the probe. In further experiments, the probe will be directed towards other ICs (e.g., the communication unit's CPU, the LAN controllers, etc.).

Figure 15 shows the special set-up used for the EMI fault-injection experiments. The burst-generator is controlled by the line AL7 of the output latch of the comparator node. Also the power supply of the node under test is controlled by the comparator node; the AL6 line of the output latch is used for this purpose. Monitoring and collecting data of the experiments, and if necessary reloading of the node subjected to fault-injection is performed by the workstation. The data obtained by the workstation can subsequently be evaluated according to the defined measures.

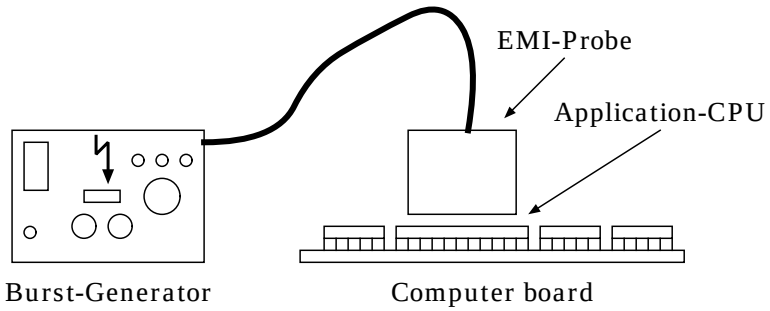

Figure 14: Coupling

7 Comparison of the Fault Injection Techniques

This section presents a comparison of the three fault injection techniques taking into account

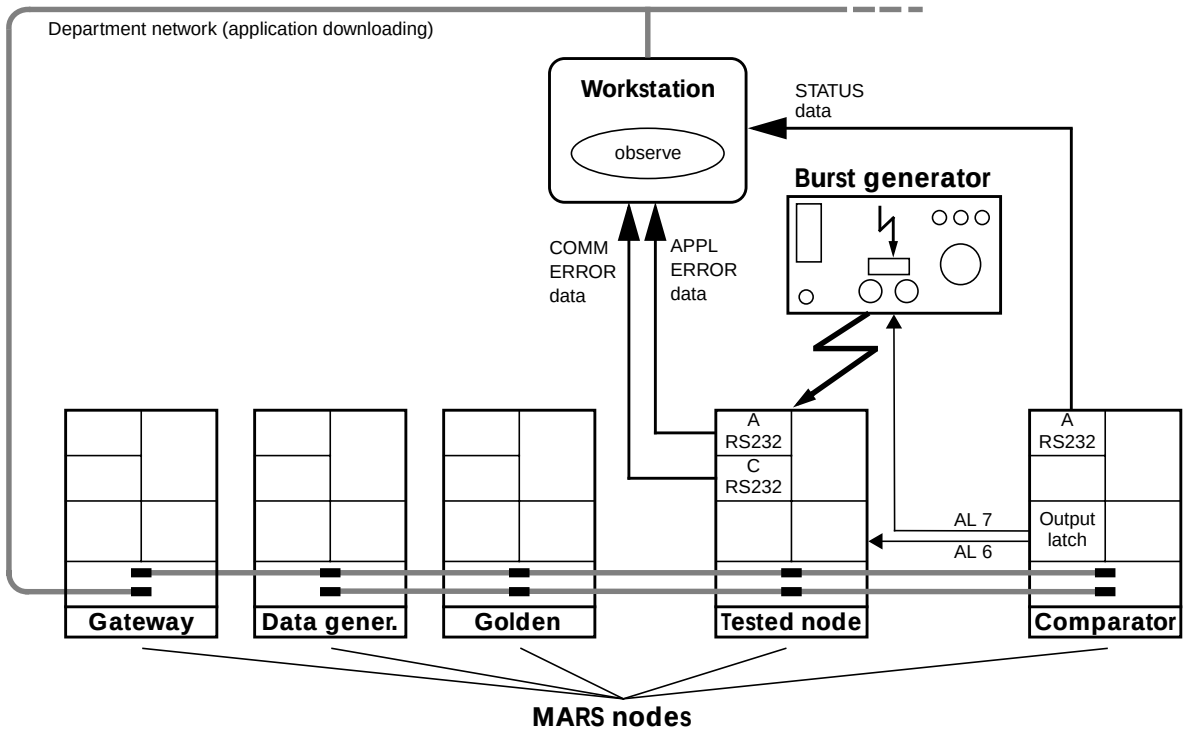

Figure 15: Experimental set-up for EMI

their fundamental and practical limitations. The comparison is based on five attributes describing the characteristics of the fault injection techniques: *controllability*, with respect to *space* and *time*, *flexibility, repeatability* and *physical reachability.*

A characterization of the fault injection techniques based on these attributes is shown in Table 1 and explained below. For each attribute, the fault injection techniques are graded on the scale *none, low, medium* and *high.* Note that this comparison does not consider the actual impact of the injected faults, i.e., the type of errors that are produced. We will also compare the possibilities to conduct latency measurements for the three methods.

Controllability

We consider controllability with respect to both the *space* and *time* domains. (The space domain corresponds to controlling *where* faults are injected, while the time domain corresponds to controlling *when* faults are injected.)

Table 1: Characterization of fault injection methods

Attributes	Heavy-ion	Pin-level	EMI
Controllability, space	low	high	low
Controllability, time	none	high/medium	low
Flexibility	low	medium	high
Reproducibility	medium[1]	high	medium[1]

Table 1: Characterization of fault injection methods

Attributes	Heavy-ion	Pin-level	EMI
Physical reachability	high	medium	medium

[1.] Reproducibility will be investigated in this study

Heavy-ion radiation has low controllability for the space domain. Faults are confined to the irradiated circuit. Faults can also be confined to specific blocks of a circuit, if the rest of the circuit is shielded. However, shielding will not be used in this study. The time of the injection of a fault cannot be controlled as the decay of the Cf-252 source is governed by a random process.

Pin-level fault injection has high controllability in both the space and time domain. Note, however, that timing controllability may be hampered by the problem of synchronizing the fault injection with the activity of the system, especially when the clock frequency of the target system is high.

EMI fault injection has low controllability in the space domain. The probe may be directed to a specific circuit, but faults may be injected also in surrounding circuits. The time of injection can be synchronized with system activity, but it is difficult to determine exactly when a fault is injected.

Flexibility

Flexibility here refers to the possibility and ease of changing the target circuit in an experimental set-up.

Heavy-ion radiation has rather low flexibility as the preparation for each IC type involves several steps including opening the circuit package, mechanical and electrical adaptation between the target system and miniature vacuum, and development of a comparator card (if latency measurements are required). Because of the effort involved in the preparations, heavy-ion experiments are typically conducted only for a few highly integrated key components in a system.

Pin-level fault injection can achieve a high degree of flexibility provided a comprehensive fault injection tool, such as MESSALINE, is used. However, flexibility may be restricted by the difficulty to physically access the pins of modern IC packages, and problems caused by the extra load capacitances introduced by the connection probes.

EMI fault injection has high flexibility as there is no physical connection between the target circuit and EMI probe.

Reproducibility

Reproducibility refers to the ability to reproduce results statistically for a given set-up and/or repeat individual fault injections exactly. Statistical reproducibility of results is an absolute requirement to ensure the credibility of fault injection experiments. The possibility to repeat

experiments exactly, or at least with a very high degree of accuracy, is highly desirable, particularly when the aim of the experiments is to remove potential design/implementations faults in the fault tolerance mechanisms.

Heavy-ion radiation experiments cannot be repeated exactly due to the lack of controllability. Previous research have shown that results are statistically reproducible among different specimens of specific ICs. The reproducibility will be investigated also for the target circuits in this study.

For **pin-level fault injection** it is possible to accurately reproduce the injection of a selected fault with MESSALINE. However, although reproducing an experiment is not a major problem for a centralized control automaton, this is not always achievable in the case of a distributed architecture.

EMI fault injection experiments may be sensitive to small changes in the positioning of the probe. What impact this might have on the statistical reproducibility of EMI results remains to be investigated. One of the important objectives of this study is indeed to access the reproducibility of EMI injection. Exact repeatability of fault injection is not possible for EMI.

Physical reachability

Physical reachability is the ability to reach possible fault location (nodes) in a system

Heavy-ion radiation has high physical reachability as faults are injected in internal locations in ICs.

Pin-level fault injection has varying physical reachability depending on the level of integration for the target system. Physical reachability is low for highly integrated systems consisting only of a few VLSI circuits. For the MARS system, which uses a mixture of VLSI, LSI, MSI and SSI circuits, physical reachability is rather high.

EMI fault injection has similar physical reachability as pin-level injection as most faults probably 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.

An important aspect in the comparison of the three fault injection techniques is the possibility to measure error detection latency. For the heavy-ion technique such measurements rely on the use of the golden chip technique. This requires that the target IC is operated synchronously with a reference IC. However, this may not be possible for ICs with non-deterministic external behaviour (caused, for example, by a varying number of wait states cycles inserted during memory accesses).

Latency measurements does not pose a problem for pin-level injection, as the time of the injection of a fault is explicitly known. For EMI fault injection, latency measurements are difficult. In principle, the golden chip techniques could be used also in this case. However, a major problem is to confine the disturbances to the target circuit. In this study, EMI will therefore not be used for latency measurements.

8 Expected Achievements

In this section we discuss the expected achievements of this study. We also highlight some of the difficulties involved in comparing the results from the three fault injection techniques.

As previously mentioned, there are two main objectives of the study: (i) to compare the impact and features of the three fault injection techniques, and (ii) to validate the fault tolerance mechanisms incorporated into the MARS system.

The goal for the comparison of the fault injection techniques is to identify similarities and differences in the error sets generated by the three techniques. If the error sets are found to be disjoint the fault injection techniques can be judged as fully complementary. In this case, applying all three techniques in the validation of a fault tolerant system would lead to significant improvement in the confidence of the results. The improvement in confidence is reduced if the error sets are found to be overlapping. To make the comparison meaningful, we will try to achieve as much similarity as possible between the three experiments by focusing the fault injection on the same ICs. Furthermore, in the case of pin-level fault injection only transient faults will be injected to achieve similarity with the heavy-ion and EMI techniques.

To compare the impact of injected faults is not an easy task in physical experiments owing to the difficulties involved in observing the internal state of the system. For the first set of experiments using the basic experimental set-up (without a logic analyser), differences in the error sets will be determined indirectly by comparing the coverage figures for the various error detection mechanisms used in the MARS system. We will develop an error classification scheme based on the assumption that differences in the error sets will be reflected in the coverage distribution among these mechanisms. We consider this a viable approach as the MARS system uses a very large number of error detection mechanisms; the communication unit, for example, uses 13 hardware error detection mechanisms, 254 other hardware and software exceptions, 91 software assertions and 49 other software checks.

To corroborate the validity of this error classification scheme and allow a more detailed comparison of the impact of the fault injection techniques, heavy-ion and pin-level experiments will be conducted with more advanced experimental set-ups using a logic analyser. The logic analyser will observe the behaviour of the ICs subjected to fault injection as well as the hardware error detection mechanisms. Access to logic analyser data allows more sophisticated error classification schemes to be developed. The impact of the errors can also be studied in more detail; it will be possible to distinguish, for example, between control flow errors, data errors and errors without effect.

The primary goal for the validation of the MARS system is to estimate the fail-silence coverage for the MARS node and to estimate the individual coverage of the various error detection mechanisms that provide the fail-silence property. Estimation of the fail-silence coverage is indeed important, as the MARS system implements fail-silence by a combination of several rather simple error detection mechanisms whose individual coverage strongly depends on the types of errors that occur. The purpose of using three different fault injection techniques each representing widely different and distinct fault sets is to increase the confidence in the results by providing a larger set of errors for which the error detection mechanisms are tested. Of course, the confidence improvement may not be achieved if the

comparison of the fault injection techniques shows that their error sets overlap.

Measuring the coverage of the individual mechanisms will enable us to compare their effectiveness, and may also help to identify the most effective combinations of mechanisms. Several different combinations of mechanisms will be evaluated. The MARS system facilitates such experiments as most mechanisms can be disabled, either by a hardware switch or by changing the software configuration. Only the built-in error detection mechanisms of the CPU cannot be switched off.

The first set of experiments using the basic experimental set-ups will only allow estimation of asymptotic coverage figures. For the heavy-ion and pin-level techniques, the more advanced set-ups will allow us to determine the latency distributions of the error detection mechanisms. The logic analyser data can help us identify the reasons for fail-silence violations, if such violations should occur. This requires that we succeed in defining the triggering conditions for the logic analyser so that it captures those parts of the program execution where the reason for a fail-silence violation can readily be observed.

It should be noted that estimating the ‘real’ coverage of error detection mechanisms from fault injection experiments is practically impossible for most systems. The reason is that the real fault set usually is not known in detail, and even less is known about the probability of occurrence of the individual faults. In principle, an estimate of the ‘real’ coverage can be calculated as a weighted mean of the coverage factors obtained by different fault injection methods. However, the lack of knowledge about the ‘real’ faults makes it very difficult — and in many practical cases impossible — to calculate the weight factors.

Each fault injection technique should therefore be considered strictly as a “benchmark” method that can be used to evaluate the relative effectiveness of different error detection mechanisms. Combining several fault injection techniques improves the possibility to investigate coverage sensitivity with respect to changes in the error set.

9 Conclusions

This report described three physical fault injection techniques – heavy-ion radiation, pin-level fault injection, and EMI – and how they will be used to validate the MARS system. In a comparison of the fundamental and practical limitations of the three fault injection techniques, it was shown that pin-level and EMI fault injection are more flexible than the heavy-ion radiation technique, and that the pin-level technique provides higher controllability than both the EMI and heavy-ion radiation techniques. On the other hand, a unique feature of the heavy-ion technique is that faults can be injected internally in integrated circuits. One of the main results of this study will be a more comprehensive comparison of the three techniques considering also the observed impact of the injected faults.

Another goal of this study is to investigate the adequacy of the error detection mechanisms implementing the fail-silence property of the MARS node. The MARS approach to fail-silence relies on a combination of time redundancy, checksums and simple checks carried out by hardware or software. This approach is much less costly than using, for example, hardware duplication to achieve fail-silence. The evaluation of the MARS system should therefore be of great value to designers who look for cost-effective ways of achieving fail-silence.

References

[Arlat et al. 1990a] 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*, vol. 16, no. 2, pp.166-182, 1990.

[Arlat et al. 1990b] J. Arlat, M. Aguera, Y. Crouzet, J. Fabre, E. Martins and D. Powell, "Experimental Evaluation of the Fault Tolerance of an Atomic Multicast Protocol," *IEEE Transactions on Reliability*, vol. 39, no. 4, pp.455-467, 1990.

[Arlat 1992] J. Arlat. "Fault Injection for the Experimental Validation of Fault-Tolerant Systems," in *Proc. Workshop Fault-Tolerant Systems*, pp. 33-40, Kyoto, Japan, IEICE, 1992.

[Arlat et al. 1993] J. Arlat, A. Costes, Y. Crouzet, J.-C. Laprie and D. Powell, "Fault Injection and Dependability Evaluation of Fault-Tolerant Systems," *IEEE Transactions on Computers*, vol. 42, no. 8,1993.

[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)*, pp. 171-176, Vienna, Austria, IEEE, 1986.

[Damm 1988] A. Damm, *Experimental Evaluation of Error-Detection and Self-Checking Coverage of Components of a Distributed Real-Time System*, Ph. D. Thesis, Technisch-Naturwissenaschaftliche Fakultät, Technische Universität Wien, Vienna, Austria, 1988.

[Dugan and Trivedi 1989] J.B. Dugan and K.S. Trivedi, "Coverage Modeling for Dependability Analysis of Fault-Tolerant Systems," *IEEE Transactions on Computers*, vol. 38, no. 6, pp.775-787, 1989.

[Goswami and Iyer 1992] K.K. Goswami and R.K. Iyer. *DEPEND: A Simulation-Based Environment for System Level Dependability Analysis*, Tech. Report CHRC-92-11, Univ. of Illinois at Urbana-Champaign, 1992.

[Grünsteidl and Kopetz 1991] G. Grünsteidl and H. Kopetz. "A Reliable Multicast Protocol for Distributed Real-Time Systems," in *Proc. 8th IEEE Workshop on Real-Time Operating Systems and Software*, Atlanta, GA, USA, IEEE, 1991.

[Gunneflo et al. 1989] U. Gunneflo, J. Karlsson and J. Torin. "Evaluation of Error Detection Schemes using Fault Injection by Heavy-ion Radiation," in *Proc. 19th Int. Symp. Fault-Tolerant Computing (FTCS-19)*, pp. 340-347, Chicago, IL, USA, IEEE, 1989.

[Jenn et al. 1993] E. Jenn, J. Arlat, M. Rimén, J. Ohlsson, and J. Karlsson, "Fault Injection into VHDL Models: The MEFISTO Tool," in *Proc. 24nd. Int. Symp. on Fault-Tolerant Computing (FTCS-24)*, pp. 336-344, Austin, TX, USA, IEEE, 1994

[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.

[Kanawati et al. 1992] G.A. Kanawati, N.A. Kanawati and J.A. Abraham. "FERRARI: A Tool for the Validation of System Dependability Properties," in *Proc. 22nd. Int. Symp. on Fault-Tolerant Computing (FTCS-22)*, pp. 336-344, Boston, MA, USA, IEEE, 1992.

[Karlsson et al. 1994] J. Karlsson, P. Lidén, P. Dahlgren, R. Johansson, and U. Gunneflo. "Using Heavy-ion Radiation to Validate Fault-Handling Mechanisms," in *IEEE Micro*, vol. 14, no. 1, pp. 8-23, 1994.

[Kopetz et al. 1989] H. Kopetz and G. Grünsteidl and J. Reisinger, ”Fault-Tolerant Membership Service in a Synchronous Distributed Real-Time System,” in *Proc. 1st Int. Working Conf. on Dependable Computing for Critical Applications (DCCA-1)*, ISBN 3-211-82249-6, Springer Verlag, pp. 197-212, Santa Barbara, CA, USA, August 23-25, 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, Vienna, Austria, 1991.

[Kopetz 1992] H. Kopetz, G. Fohler, G. Grünsteidl, H. Kantz, G. Pospischil, P. Puschner, J. Reisinger, R. Schlatterbeck, W. Schütz, A. Vrchoticky, and R. Zainlinger. The Distributed, Fault-Tolerant Real-Time Operating System MARS. IEEE Operating Systems Newsletter, vol. 6, no. 1, 1992.

[Madeira et al. 1991] H. Madeira, P. Furtado e Joao and J.G. Silva. *RIFLE: A General Purpose Fault Injector System,* Research Report DEE-UC-007-91, 1991.

[Miremadi et al. 1992] G. Miremadi, J. Karlsson, U. Gunneflo and J. Torin, “Two Software Techniques for On-line Error Detection”, in *Proc. 22nd Int. Symp. on Fault -Tolerant Computing (FTCS-22)*, pp. 328-335, Boston, MA, USA, IEEE, 1992.

[Philips Semiconductors 1991]. Philips Semiconductors, *SCC68070 User Manual 1991, Part 1 — Hardware*, 1991.

[Powell et al. 1988] D. Powell, G. Bonn, D. Seaton, P. Verissimo, and F. Waeselynck, “The Delta-4 Approach to Dependability in Open Distributed Computing Systems,” in *Proc. 18th International Symposium on Fault-Tolerant Computing Systems (FTCS 18)*, pp. 246 - 251, IEEE, Tokyo, Japan, IEEE, 1988.

[Powell et al. 1993] D. Powell, E. Martins, J. Arlat and Y. Crouzet. “Estimators for Fault Tolerance Coverage Evaluation,” in *Proc. 23rd Int. Symp. on Fault-Tolerant Computing (FTCS-23)*, pp. 228-237, Toulouse, France, IEEE, 1993.

[Puschner and Koza 1989] P. Puschner and C. Koza. “Calculating the Maximum Execution Time of Real/Time Programs,” in *Real-Time Systems Journal*, vol. 2, no. 1, pp. 159-176, Sep., Kluwer Academic Publishers, 1989.

[Reisinger 1992] J. Reisinger, “Time Driven Operating Systems - A Case Study on the MARS Kernel,” in *Proc. 5th ACM SIGOPS European Workshop*, Le Mont Saint Michel, France, Sep. 1992.

[Reisinger 1993] J. Reisinger, *Konzeption und Analyze eines zeitgesteuerten Betriebssystems für Echtzeitsanwendungen*, Ph. D. Thesis, Technisch-Naturwissenaschaftliche Fakultät, Technische Universität Wien, Vienna, Austria, 1993

[Reisinger and Steininger 1994] J. Reisinger and A. Steininger, “The Design of a fail-Silent processing Node for MARS” in *Distributed Systems Engineering Journal*, Institution of Electrical Engineers, 1994. To appear.

[Schaffner Instruments] *User’s Manual: Fast Transient / Burst Generator NSG 1025.*

[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)*, pp. 138-143, Vienna, Austria, IEEE, 1986.

[Segall et al. 1988] Z. Segall, D. Vrsalovic, D. Siewiorek, D. Yaskin, J. Kownacki, J. Barton, D. Rancey, A. Robinson and T. Lin. “FIAT — Fault Injection based Automated Testing Environment,” in *Proc. 18th Int. Symp. on Fault-Tolerant Computing (FTCS-18)*, pp. 102-107, Tokyo, Japan, IEEE, 1988.

[Steininger and Reisinger 1991] A. Steininger and J. Reisinger. “Integral Design of Hardware and Operating System for a DCCS, in “*Proc. 10th IFAC Workshop on Distributed Computer Control Systems*,” Semmering, Austria, 1991.

[Vrchoticky 1992] *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,* vol. 39, no. 4, pp.492-499, 1990.