

Using Heavy-Ion Radiation to Validate Fault-Handling Mechanisms

Fault injection is an effective method for studying the effects of faults in computer systems and for validating fault-handling mechanisms. Our approach involves injecting transient faults into integrated circuits by using heavy-ion radiation from a Californium-252 source. The proliferation of safety-critical and fault-tolerant systems using VLSI technology makes such attempts to inject faults at internal locations in VLSI circuits increasingly important.

Johan Karlsson

Peter Lidén

Peter Dahlgren

Rolf Johansson

*Chalmers University of
Technology*

Ulf Gunneflo

Carlstedt Elektronik AB

One of the most critical issues in designing a fault-tolerant computer is the validation of its fault-handling mechanisms. These mechanisms include those parts of the system that perform such functions as error detection, error masking, recovery, and system reconfiguration. Ineffective or unintended operation of these mechanisms can significantly impair the dependability of a fault-tolerant computer system. Even a small reduction of fault coverage may significantly degrade system dependability. At the same time, the growing complexity of computer systems makes the validation of fault-handling mechanisms increasingly difficult.

Analytical dependability modeling and formal design verification are important tools for validating fault-tolerant computers. However, when we validate fault-handling mechanisms, we must also deal with faults and their impact on the system—that is, with both the errors and the error propagation process. These aspects of a computer system are very hard to model by analytical techniques. Consequently, we must also use experimental techniques for validating a fault-tolerant system. The obvious way to experimentally validate fault-handling mechanisms is to inject faults into either a prototype or a software simulation model of a system.

Ongoing developments in integrated circuit technology make it increasingly difficult to phys-

ically inject realistic faults into a prototype or an actual system. Previously, when designers used small- or medium-scale integration circuits, faults affecting bonding wires and pin connections predominated. We could easily imitate the effects of such faults by introducing faults via probes connected to the pins of the integrated circuits. Due to the modest complexity of the circuits, pin-level fault injection also lets us mimic the effects of internal faults with reasonable accuracy. However, the advent of VLSI technology has dramatically changed the picture. A much larger proportion of the faults now occur internally in the chip, and mimicking the effects of many of these faults by pin-level faults is difficult indeed.

Here, we discuss a fault-injection method that, at least partially, solves the problem of injecting faults at internal locations in VLSI circuits. Our approach involves using heavy-ion radiation from a Californium-252 source for injecting single-event upsets, or bit flips, into integrated circuits. Our fault injection experiments with the MC6809E microprocessor will help illustrate this technique. These experiments investigated the error behavior of the microprocessor when subjected to heavy-ion radiation. They also evaluated the effectiveness of several error detection mechanisms based on the monitoring of the behavior of the microprocessor. We will also show how simulation-based fault injection can

help explain observations made in physical fault injection experiments.

In general, fault injection addresses two aspects of the validation process: fault forecasting and fault removal.¹ Fault forecasting serves to rate the efficiency of the fault-handling mechanisms—to estimate measures such as error detection coverage or error detection latency. Such measures are important parameters in analytical dependability models. Fault removal is the process of removing and identifying unexpected deficiencies (design faults) in the fault-handling mechanisms. Regarding the exact meaning of the words "fault," "error," and "failure," we adhere to the now widely recognized dependability terminology proposed by Working Group 10.4 on Dependable Computing and Fault-Tolerance of the International Federation for Information Processing.¹ This terminology defines "fault" as the cause of an error, while "error" is the manifestation of a fault as reflected by the system state. An error may or may not lead to a "failure," which is the interruption of the service delivered by a system.

Overview of fault injection techniques

Over the years, many researchers have addressed the problem of validating fault-handling mechanisms experimentally. Numerous papers on validation of fault-tolerant systems or individual fault-handling mechanisms and on tools for fault injection have been published. Fault injection techniques fall into two main categories:

- techniques for injecting faults into software-simulation models of systems, and
- techniques aimed at injecting faults into physical systems, either prototypes or actual systems.

These two categories complement each other as they can serve in different phases of the design process.

Simulation-based fault injection. Simulation-based fault injection can be used at various levels of abstraction such as circuit, gate, register-transfer, or system. Circuit-level simulations let us study hardware fault mechanisms and validate fault models used at the gate level or register-transfer level. Gate-level or register-transfer-level simulations enable us to validate hardware error detection mechanisms, such as used in a CPU. System-level simulations are appropriate for validating more complex mechanisms, such as fault-tolerant communication protocols or recovery procedures. As we will describe, we have used simulation-based fault injection both at the circuit-level and the register-transfer level to explain phenomena observed in physical experiments with the heavy-ion technique.

Simulation-based fault injection has several advantages compared to injecting faults into a physical system. Perhaps most importantly, simulation-based fault injection can be used early in the design process, which facilitates early detec-

Most real faults cannot be injected physically. Instead, we must select a set of injectable artificial faults believed to be representative of the real faults.

tion of design faults, thus reducing the cost for correcting such faults. It also provides almost perfect controllability and observability, as we can freely select both the time and location for injecting faults as well as the observation points for tracing the system response. The main drawback to simulation-based fault injection is the overhead time involved in simulations, which puts practical limitations on the amount of hardware and system activity we can simulate.

Fault injection in physical systems. Because of the complexity involved, we cannot validate a fault-tolerant system by simulation-based fault injection alone. We must therefore provide techniques for injecting faults into physical systems, that is, prototypes or actual systems. Before describing the various techniques used for injecting faults into physical systems, we address some fundamental issues concerning the use of such techniques for validating fault-handling mechanisms.

A general problem in this context is that most real faults such as oxide breakdowns and open metal wires are impossible to inject physically. Instead, we must select a set of injectable artificial faults believed to be representative of the real faults in that they cause errors which are similar to those caused by the real faults. Validating the representativeness of the injected faults for the real faults is a very difficult task because the knowledge about the real faults in most cases is limited, particularly before a system has been put to use.

Using software simulations to compare the errors caused by the injectable faults and the real faults is one way to qualitatively validate the representativeness of the injected faults. Accurate simulation models must, of course, be available for both fault sets. However, such validation is often too time consuming and expensive in practice. Instead, we select the injected faults to achieve a large variation in the error patterns. We then hope that these error patterns are more difficult to detect than the error patterns real faults cause, and that the validation therefore in some sense is pessimistic.

We can divide the techniques used for injecting faults into physical systems into four major categories:

- pin-level fault injection,
- built-in fault injection,
- software-implemented fault injection, and
- external disturbances.

Pin-level fault injection. By the 1950s, researchers had already begun using this approach for generating fault dictionaries for system diagnosis; since then it has been used in numerous studies in industry and academia.^{2,3} To employ pin-level injection effectively requires using a tool such as Messaline² that allows full automation of fault injection experiments. Messaline uses a Macintosh II computer to control special hardware modules responsible for fault injection, to control the target system, and for data collection. Messaline supports two different pin-level techniques: forcing and insertion. Forcing involves injecting faults by forcing the pins to an erroneous voltage level. The insertion technique entails removing the integrated circuit from its socket and inserting it in a box where extra logic manipulates both input and output signals. By combining forcing and insertion Messaline can inject a variety of fault types including stuck-at zero, stuck-at one, physical bridging, logical bridging, and logical inversion.

In principle, pin-level fault injection can provide full control of the injection of faults. However, many pin-level fault injecting tools—including Messaline—do not allow full controllability, as the injection of faults cannot be synchronized with the system clock. In fact, using pin-level injection is a major challenge in modern systems that use clock frequencies of, say, 50 MHz or higher. Here, problems arise in synchronizing fault injection with system activity and in designing connections probes that do not disturb the normal operation of the system. The main disadvantage to pin-level fault injection arises from the difficulty of injecting faults that generate error patterns that are representative of the error patterns caused by faults occurring internally in integrated circuits. As packing density of integrated circuits increases, this problem becomes more severe.

Built-in fault injection. This approach involves incorporating special fault injection hardware into the system. In modern designs using VLSI technology this means that we must incorporate fault injection mechanisms into the integrated circuits. In this case, the fault injection hardware could conveniently combine with scan chain techniques used for circuit testing. Built-in fault injection has successfully increased the efficiency and quality of system test for mainframe computers.⁴

Software-implemented fault injection. This method introduces faults—or rather errors, the manifestations of faults—in memory locations or CPU registers by means of software.^{5,6} The underlying idea is that software could emulate the error behavior caused by most faults, for example, by altering bits in instructions, address pointers or data, or

by changing whole sequences of instructions. Software-implemented fault injection provides better control of the injection of faults than pin-level injection, and is less expensive than built-in fault injection. However, its ability to model certain types of faults, such as those affecting the control section of a CPU, has not yet been fully investigated. Also, the execution of the software responsible for injecting the faults affects the timing characteristics of the system, which may prevent testing of time-critical functions.

External disturbances. This approach can enable us to reach those parts of an integrated circuit that pin-level or software-implemented fault injection cannot reach, such as the control section of a CPU. Investigations along this line focus mainly on two types of disturbances: the heavy-ion technique we discuss, and the use of power supply disturbances. Both techniques make it possible to inject faults whose effects are difficult to imitate by other fault injection techniques. However, a lack of controllability impairs these techniques; only small variations in the injected fault set can be achieved. Therefore, they serve strictly as complements to the other techniques just described.

We can inject power supply disturbances by introducing short voltage sags on the power supply line of the target circuit. This method facilitated investigations on the effect of power supply disturbances on the Intel 8251A USART⁷ and the Motorola MC6809E 8-bit microprocessor.⁸ The results show that the voltage sags affected the control section of these circuits by causing transitions in the operating state of the circuits. However, the MC6809E experiment showed that the voltage sags were extremely unlikely to cause bit errors in the user registers, and that the error behavior varied considerably between different specimens of the circuit. The latter suggests that reproducing results from fault injection experiments may be difficult using power supply disturbances because of a high sensitivity to small changes in the experimental setup.

Using heavy-ion radiation from Cf-252 for fault injection

The major advantage of using heavy-ion radiation for fault injection is that it lets us inject faults into VLSI circuits at locations that pin-level or software-implemented fault injection cannot reach. In most cases, the effects of the faults injected by heavy-ion radiation are bit flips in the information stored in memory elements (flip-flops and latches). Such bit flips are also called soft errors, as the heavy ions do not permanently damage the hardware. (Permanent damage may occur from particle-induced latch-ups commonly found in complementary metal-oxide semiconductor circuits, a problem we discuss later.) A single heavy ion usually affects only one information bit, although multiple bits sometimes may be affected. Therefore, single-bit errors cause most errors seen on the output pins of an irradiated circuit, provided the

heavy-ion flux is adjusted to an appropriate level. That is, the flux must be adjusted to a level for which the mean time between errors (heavy-ion hits) is much longer than the mean error propagation time.

Generating single-bit errors gives the heavy-ion method an advantage from the standpoint of validation. Single-bit errors arguably have more subtle effects than multiple-bit errors, as they affect less information. Consequently, the effects of single-bit errors usually should be more difficult to detect than the effects of multiple-bit errors. Of course, this is not true for all types of error detection mechanisms, as for example when parity codes or other forms of coding protect information. Still, it is a valid argument for many system-level error detection mechanisms such as the watchdog mechanisms we will evaluate later. Note, though, that single-bit errors such as those occurring internally in a microprocessor are often transformed into multiple-bit errors during the error propagation process.

Another advantage is that faults are reasonably spread out within a circuit, as most VLSI circuits have many sensitive memory elements. Thereby, the injected faults generate a variety of error patterns, allowing a thorough testing of fault-handling mechanisms.

The idea of using heavy-ion radiation from Cf-252 for fault injection originally arose as a method for assessing the single-event upset vulnerability of circuits considered for space applications.⁹ The term "single-event upset" means a soft error—one or several bit flips—caused by a single ionizing particle. In space, heavy-ions found in the cosmic ray spectrum cause SEUs. The method emerged as an inexpensive way of measuring the SEU cross section for integrated circuits. The SEU cross section is a measure of the SEU vulnerability of a circuit. Such measurements usually take place in particle accelerators, making them quite expensive.

Designers use the SEU cross section together with data on the radiation environment in space to predict SEU rates for onboard electronic equipment. The SEU cross section is also of interest when using heavy-ion radiation for validating fault-handling mechanisms. To assess the feasibility of using heavy-ion radiation for a particular system, we must first determine the cross section for potential target circuits before undertaking the actual fault injection experiments. The SEU and latch-up cross section box (next page) further explains the SEU cross section and how it can be measured.

Fault mechanism. 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 current pulse in the associated circuits. If the particle hits a sensitive region in a memory element, the current pulse will change the state of that memory element. The sensitivity of memory elements varies, depending on their design. In some designs bit flips may occur in both direc-

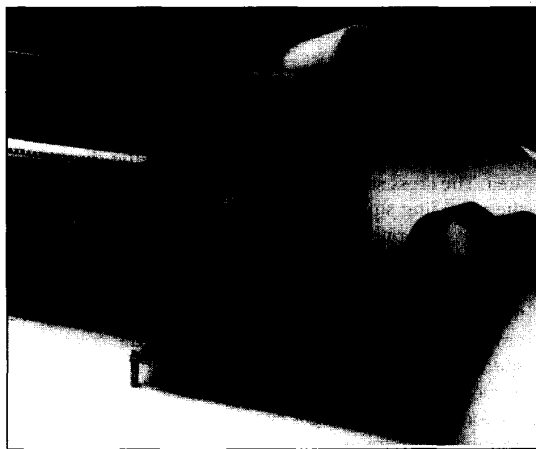


Figure 1. The experimental setup.

tions, while in other designs they occur only in one direction. The latter is true for the user registers in the MC6809E microprocessor in which bit flips occurred only from 1 to 0. As described in the Circuit-level fault injection box (page 14), simulation-based fault injection at the circuit level can determine the reason for such asymmetric behavior. Results from cross-section measurements show that circuits manufactured in NMOS, CMOS, and bipolar technology may be sensitive to heavy-ion radiation. These results show that SEU sensitivity increases with packing density, but also that it may vary considerably for different circuit types.

Miniature vacuum chamber. Irradiation of circuits must take place in a vacuum, as air molecules attenuate the heavy ions. Therefore, for irradiating circuits in a system, we have designed a miniature vacuum chamber containing the Cf-252 source and the irradiated circuit (see Figure 1). So that the chamber plugs directly into the socket normally holding the target circuit, the circuit's pin connections extend through the bottom plate of the miniature vacuum chamber. The distance between the irradiated circuit and the Cf-252 source is approximately 35 mm; it varies slightly depending on the circuit package. Inserting an extension tube between the bottom plate and the rest of the miniature vacuum chamber increases this distance. That way, we can reduce the heavy-ion flux impinging onto the chip. The vacuum chamber also contains an electrically maneuvered shutter, used for shutting the radiation on and off. To prevent radioactive contamination, the vacuum chamber must be assembled in a radiologically controlled area under a vented hood. After pumping out the air and disconnecting the vacuum pump, technicians can handle the miniature vacuum chamber freely, without making

Continued on page 13

SEU and latch-up cross section

In the context of computers, cross section is a measure of how sensitive semiconductor circuits are to radiation. Traditionally, cross section describes a nuclear reaction probability. As the name indicates, it has the dimension area. The cross section is the total sensitive area for which the incoming studied particle is not transparent. In our context, this phenomena deals with heavy ions or other radiation, causing single-event upsets or latch-up in a semiconductor circuit. Researchers traditionally use cross-section values for predicting upset probabilities for the space environment, where heavy ions found in the cosmic ray spectrum may cause SEUs and latch-ups in onboard electronic equipment. Determining the SEU and latch-up cross section of potential target circuits is also a necessary and important step for assessing the feasibility of using heavy-ion radiation for validation of a fault-tolerant system.

We calculate the cross section as the event rate (measured in events per second) divided by the heavy-ion flux (measured in ions per second and centimeter squared). Here, an event is the detection of an SEU or a particle-induced latch-up. (Latch-up is common in CMOS devices). We can measure the SEU cross section in two ways: either as the register-bit cross section or the application cross section. The register-bit cross section is a measure of the SEU sensitivity for one single-memory element. It is a workload-independent measure of the SEU sensitivity.

The application cross section is a measure of the total SEU cross section for the device when it is performing a characteristic task. In this case, most memory elements in a device have a duty cycle of less than 100 percent, which means that some of the SEUs become overwritten and thus do not contribute to the measured cross section. In this context the duty cycle denotes the fraction of the time a memory element holds a value that later will be used. The application cross section varies for different workloads depending on how the device is used.

The register-bit cross section is a generic measure of the SEU sensitivity, and is therefore a more convenient measure than the application cross section. We can easily determine the register-bit cross section for memory devices such as RAM, EPROM, and EEPROM circuits. We can also determine it for user registers in microprocessors. However, many registers in a microprocessor are not directly accessible to the user, making the register-bit cross section difficult or impossible to measure. The application cross section takes account of the contribution such registers make to the SEU sensitivity. Consequently, the application cross section is an important complement to the register-bit cross section for characterizing the SEU sensitivity of complex circuits such microprocessors.

Table A shows the SEU register-bit cross section for a few circuits. Nichols et al.¹ and Harboe-Sorensen et al.² provide collections of cross-section values for many semiconductor devices. For the latch-up cross section, there is just one measure for the entire device.

How to induce desired single-event effects. Researchers usually make cross-section measurements by irradiating the target device in a particle accelerator. For a particle to induce a single-event effect, it must have a sufficient linear energy transfer. The LET value determines the amount of charge the particle generates when it penetrates the device. If the LET is under a threshold value, the deposited charge is not sufficient to cause a single-event effect. Above the LET threshold the cross section increases with increasing LET up to a saturation LET value. Above the saturation value an increase in LET does not cause any further increase of the cross section. In an accelerator we can control the LET of the particles by using different types of ions such as krypton, neon or nitrogen, and by varying the angle from which the ion beam hits the device. In this way, we can determine both the threshold LET and the saturation LET for a device.

If we only need to know the saturation cross section, it is not critical to use ions of a well-defined LET, as long as the LET exceeds the saturation value. The fission fragments emitted from Californium-252 have a mean LET of 43 MeV $\text{mg}^{-1}/\text{cm}^2$, which is above the saturation LET for most non-radiation-hardened circuits. Using a Cf-252 source therefore provides an inexpensive alternative to accelerator tests for measuring the saturation cross section. Note, however, that some Cf-252 experiments may give cross-section results that differ from values obtained by accelerator experiments.³

How to measure the cross section. The cross section is measured by determining the heavy-ion flux and the single-event rate. We can measure the heavy-ion flux by replacing the target device with a silicon surface barrier detector connected to a conventional nuclear spectroscopy measurement system. We can measure the single-event rate in several different ways depending on the type of circuit being tested.

Measuring the SEU rate when determining the register-bit cross section for memory devices is straightforward. First, we load a known bit pattern into the device. Then we continuously read memory contents during the irradiation and compare them to those originally stored. A mismatch in the comparison indicates the occurrence of an SEU. The time between the SEU occurrences determines the SEU rate. To avoid inaccuracies in the measurements, note that the time required to examine all memory ele-

SEU and latch-up cross section (continued)

ments should be much smaller than the mean time between errors. Continuously reading the memory cells—rather than just counting the number of SEUs in the device after the irradiation period—lets us distinguish the SEUs affecting multiple bits from SEUs affecting only single bits.

The following test sequence measures the register-bit cross section for the user registers in microprocessors. The processor first executes a short program that loads a specific bit pattern into the user registers and is halted. (Typical bit patterns are all bits set to 1 or 0, or alternating 0s and 1s.) After halting for one second, the processor is reset and starts to execute a program that compares the contents of the user registers with a copy of the original contents that is stored in EPROM. If it detects a mismatch, the processor dumps the contents of the user registers via a serial line to a data acquisition computer. If no error occurs, the processor again loads the bit pattern into the user registers. It then halts, and a new comparison is made after one second. This method measures the time between errors with a resolution of one second. With it, we can measure the average register-bit cross section as well as the cross section for individual bits by examining the register dump. We used this method for measuring the register-bit cross section for two of the microprocessors in Table A.

Unfortunately, the method suffers from the "to measure is to disturb" problem, as SEUs and latch-ups occurring during the comparison and the error reporting may disturb the experiment. Ideally, the time between the comparisons should be much shorter than the mean time between errors and much longer than the time needed for the processor to make the comparison. Usually, the comparison takes less than 500 μ s for modern microprocessors; we adjust the mean time between errors to be in order of minutes. (We can control the mean time between errors by adjusting the heavy-ion flux with the distance between the Cf-252 source and the circuit.) Using one second between the comparisons is a compromise between measurement accuracy and the time needed to conduct an experiment, which basically is determined by the heavy-ion flux.

Another problem is that SEUs may generate spurious interrupts both during program execution and when the processor is halted. Consequently, all interrupt vectors must be properly initiated so that the occurrence of such interrupts gets reported adequately to the data acquisition computer.

special arrangements for radiation protection.

Opening packages. Because the packaging material also effectively stops the heavy ions, the packages of irradiated

Table A. SEU register-bit cross sections for several circuits.

Device	Register-bit cross section ($10^{-6}/\text{cm}^2$)
Intel 8086 CPU, NMOS*	5
Motorola 68000 CPU, NMOS**	10
Motorola 68020 CPU, CMOS**	5
Inmos, T414 Transputer, CMOS*	2
Hitachi HM6116 2Kx8 SRAM, CMOS*	1

*From Harboe-Sorensen et al.²

**Based on our own measurements.

We can determine the application cross section for a microprocessor by measuring the time between errors using a golden chip device, as we describe elsewhere. Koga et al.⁴ describe several other techniques for SEU rate prediction for microprocessors. To prevent latch-ups from permanently damaging CMOS circuits, we should use a current guard to turn the power off whenever latch-ups occur during testing. We can determine the latch-up rate, which is application independent, by dividing the number of current guard alarms with the total experiment time.

References

1. D. K. Nichols et al., "Update on Parts SEU Susceptibility From Heavy Ions," *IEEE Trans. Nuclear Science*, Vol. 38, No. 6, IEEE, Piscataway, N.J., 1991, pp. 1529-1539.
2. R. Harboe-Sorensen, L. Adams, and T.K. Sanderson, "A Summary of SEU Test Results Using Californium-252," *IEEE Trans. Nuclear Science*, Vol. 35, No. 6, IEEE, 1988, pp. 1622-1628.
3. R. Velazco et al., "Comparison Between Californium and Cyclotron SEU Tests," *IEEE Trans. Nuclear Science*, Vol. 36, No. 6, IEEE, 1989, pp. 2383-2387.
4. R. Koga et al., "Techniques of Microprocessor Testing and SEU-Rate Prediction," *IEEE Trans. Nuclear Science*, Vol. 32, No. 6, IEEE, 1985, pp. 4219-4224.

circuits must be opened. Opening a ceramic package with a metal lid is easily done just by removing the lid. Opening ceramic packages without metal lids requires that we remove

Circuit-level fault injection

Heavy-ion fault-injection experiments show that bit flips in the user registers of the MC6809E microprocessor occur only in one direction, from 1 to 0. To explain this asymmetric sensitivity of an NMOS register cell, we conducted a fault-injection experiment using the circuit-level simulation program SPICE. Figure A shows the design of an NMOS register cell. The stored data q is fed back through M_2 and amplified to prevent decay of the stored charge. To drive the storage node to a desired value, the resistance of M_2 must be much larger than the effective channel resistance of the pass transistor M_1 .

Only reverse-biased PN junctions are sensitive to heavy-ion-induced charge deposition. As the P type substrate of the NMOS transistors is connected to ground, any transistor drain region assigned a high voltage value will produce a reverse-biased PN junction between the drain and substrate. Which of the drain regions in the circuit are sensitive varies dynamically during operation depending on the state of the register cell.

Consider the case in which a 1 is stored in the register cell ($q=1$ and $\phi=0$). The nodes n_1 and n_3 are high and node n_2 is low, which results in the PN junctions j_1 , j_3 , j_4 , and j_5 being sensitive to charge deposition. A transient $V(n_1)$ at node n_1 caused by the charge deposition would be amplified by the chain of inverters and thus could affect the storage node n_3 . If the duration of the transient is long enough

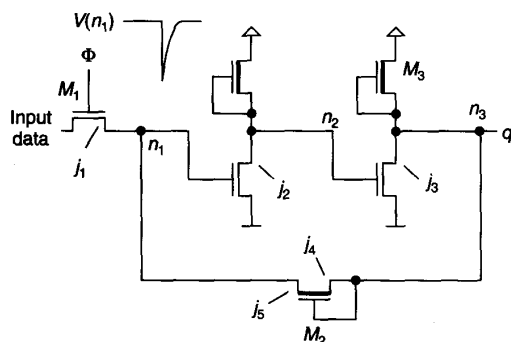


Figure A. Static NMOS register cell with a transient at node n_1 .

compared to the time delay of the inverter chain, the state of n_3 will change from high to low before the voltage at n_1 on its rising edge reaches the inverter threshold voltage, thus preventing n_1 from resuming its old value. In this case, both n_1 and n_3 have changed state; that is, a bit flip has occurred. A transient at node n_3 , conversely, cannot affect node n_1 . As the time constant through M_2 to n_1 is large, a transient of realistic duration at n_3 will not drive n_1 low before it is overwritten by M_3 .

When a 0 is stored in the cell, nodes n_1 and n_3 are low and node n_2 is high. In this case, only the PN junction j_2 is sensitive. A transient at node n_2 must pass, in the same way as a transient at n_1 , through the high ohmic transistor M_2 to affect the state of the register cell. This is not possible when considering transients caused by realistic amounts of charge. Thus we expect that only transients at node n_1 will affect the state of the cell. Consequently, we could expect only bit flips from 1 to 0 owing to heavy-ion radiation for this type of register cell.

Several fairly simple circuit-level simulations illustrate the effects of charge collection in a sensitive junction of a register cell. In these, a current pulse i_p deposits a charge to the target node of fault injection. As all the sensitive nodes are high, the current source connects from the target node to ground as shown in Figure B. As described in Messenger,¹ the waveform of the current pulse could be rather complicated. For the sake of simplicity, the simulations employed only a square pulse of duration τ and amplitude I_p , corresponding to a deposited charge $Q = \tau \cdot I_p$. The necessary amount of charge to change the state of a latch, often called the critical charge Q_{crit} , depends on the process parameters of the cell and the channel resistances of M_2 and M_3 . The intent of this example is not, however, to determine the actual magnitudes but to illustrate the basic mechanism of the bit flips. We used typical

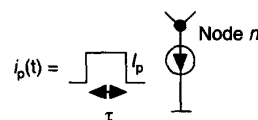


Figure B. Simulation model of charge deposition at node n .

the entire ceramic top if the pins are secured by inserting the circuit into an extra socket. For circuits with plastic packages, we must make the chip visible by etching away part of the packaging material with fuming nitric acid. We have successfully done this with several circuits without damaging

the chip or the bonding wires.

The Cf-252 source. We have used a commercially available Cf-252 source with a nominal activity of 37 kBq (1 kBq = 1,000 disintegrations per second). The source consists of a stainless steel disc with a diameter of 25 mm and thickness

Circuit-level fault injection (continued)

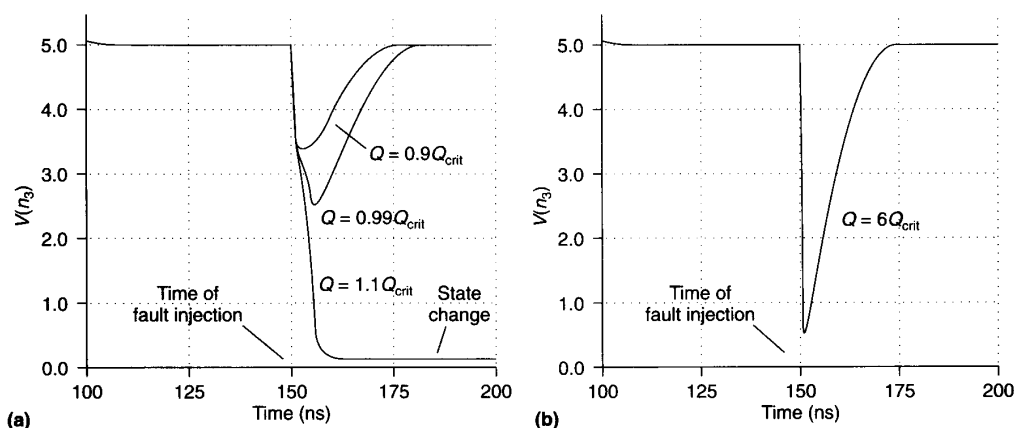


Figure C. Voltage of the storage node (n_3) when the register cell is in the high state: Fault injection at node n_1 (a); fault injection at node n_3 (b).

transistor parameters from a 3- μm process.

Figures C and D show the results of fault injections into the various sensitive nodes with the pulsewidth fixed to $\tau = 0.5$ ns. We subjected all three possible target nodes to fault injection; the figures show the resulting voltage of the storage node n_3 . The charge deposited by the current pulse is related to Q_{crit} for the plots. As Figure C shows, fault injection at node n_1 produced a register bit flip from 1 to 0 for a charge amount of $1.1Q_{crit}$. For an amount less than Q_{crit} , only a voltage glitch occurred. Fault injections into the other nodes did not result in any steady-state voltage change even though we injected a very high amount of charge.

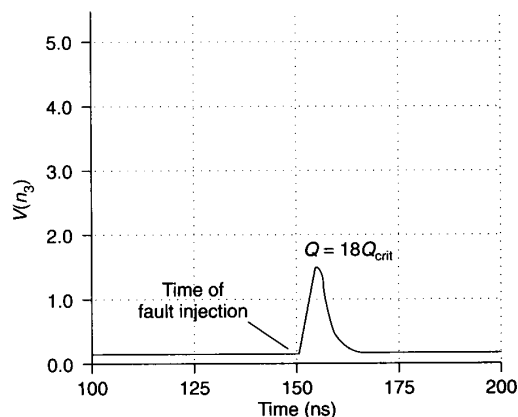


Figure D. Voltage of the storage node (n_3) when the register cell is in the low state. Fault injection occurred at node n_2 .

References

1. G.C. Messenger, "Collection of Charge on Junction Nodes from Ion Tracks," *IEEE Trans. Nuclear Science*, Vol. NS-29, No. 6, IEEE, Piscataway, N.J., Dec. 1982, pp. 2024-2031.

of 0.5 mm, onto which Cf-252 has been deposited by high temperature vacuum evaporation. The active area is 4 mm in diameter. Cf-252 decays both by alpha particle emission and by spontaneous fission. About three percent of the nuclear events generate fission fragments (heavy ions); the rest generate alpha particles. The fission fragments are of primary interest for fault injection. The charge generated by the alpha particles is too small to affect most circuits.

The fission fragments span a spectrum of mass numbers. The fission process is mainly asymmetric, meaning each fission event generates one light and one heavy particle, with average mass numbers of 106 and 142. This process produces two distinct peaks in the energy spectrum of the emitted heavy ions, one at 79 MeV for the heavy particles and one at 104 MeV for the light particles. However, the two particle groups have a similar capacity to generate charge in semi-

Table 1. Nuclear characteristics of Californium-252.

Parameter	Value
Effective half life	2.639 years
Alpha decay	96.91 %
	5.97–6.12 MeV
Spontaneous fission	3.09%
Mean fission fragment masses:	
Light particle group	106 atomic mass units
Heavy particle group	142 atomic mass units
Mean fission fragment energies:	
Light particle group	104 MeV
Heavy particle group	79 MeV
Alpha particle LET in silicon	~1.6 MeV mg ⁻¹ cm ²
Fission fragment LET in silicon	41–45 MeV mg ⁻¹ cm ²
Penetration depth for fission fragments in silicon	13–17 μ m
A 37 kBq Cf-252 source	3.6 \times 10 ⁴ alpha particles/s
emission	2.2 \times 10 ³ fission particles/s ~10 ³ neutrons/s

conductor material. As much as 95 percent of the particles have a linear energy transfer (LET) value between 41 and 45 MeV mg⁻¹ cm² in silicon, which is sufficient to cause SEUs in most nonradiation-hardened circuits.¹⁰ The LET value is proportional to the charge generation rate. The penetration depth for the fission fragments is 13 to 17 μ m in silicon. Table 1 summarizes the nuclear characteristics of Cf-252.

Use of a golden chip. Because the decay of the Cf-252 source is a random process, the heavy-ion technique does not allow us to control the time and location of the injection of faults. Not knowing when the faults are injected poses a problem when observing the system response, as no signal is available to trigger, for example, a logic analyzer. Solving this problem requires the so-called golden chip technique in which the irradiated circuit operates in lock step with an identical circuit not subjected to irradiation. Comparison of the output signals from the two circuits during each clock cycle reveals any errors. This way, an error appearing on the output pins of the irradiated circuit can immediately trigger a logic analyzer.

Thus, when using the golden chip technique, the target circuit and the golden chip both operate on a special circuit board connected to the target circuit's original socket in the circuit being validated. The special circuit board also contains the comparison logic and latches for storing the state of all pins when a mismatch is detected. Note that we cannot always employ the golden chip method. In some cases, particularly when the clocking frequency is high, extending the circuit connections may be impossible due to the extra load capacitances introduced. Furthermore, some circuits cannot operate in lock step due to asynchronous behavior.

Of course, we can use the heavy-ion technique without a golden chip device simply by irradiating a circuit and observing the fault-handling mechanisms or the behavior of the system in general. However, using a golden chip device significantly improves our ability to observe the error propagation process.

Latch-up protection. 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) working as a silicon-controlled rectifier. The excessive heat dissipation involved in a latch-up may destroy the circuit. A drastic increase in the current drawn by the circuit indicates a latch-up. To protect circuits from latch-up, we have developed a current guard that monitors the current drawn by the irradiated circuit and immediately turns off the power if the current exceeds a threshold value. We have used this current guard successfully with several CMOS circuits.

Safety precautions. Californium-252 is a highly toxic, radioactive element. Handling the Cf-252 source requires special facilities. The main safety hazard is the risk of internal contamination of the human body for the personnel who handle the Cf-252 source and the irradiated circuits that get contaminated during irradiation. Using the miniature vacuum chamber minimizes the safety hazards during the actual fault injection experiments, as the radioactive material stays inside the chamber. However, to exchange the target circuit, technicians must open the chamber in a radiologically controlled area under a vented hood using protective gloves. After assembling the chamber, they must perform a wipe test to ensure that no contamination is present on the outside of the chamber.

Limitations. Several problems arise when using the heavy-ion technique to validate fault-handling mechanisms. First, we cannot control the time and location of the injection faults. When validating fault-handling mechanisms, we must often exactly repeat a fault injection experiment, as when we want to check that corrections to the design of the fault-handling mechanisms have been effective. This is possible, at least in principle, when using other fault injection techniques such as pin-level fault injection, software-implemented fault injection or built-in fault injection. Note, though that the inherent randomness of the heavy-ion method also works to our advantage when validating systems. Because the fault set is implicitly defined and cannot be manipulated, it guarantees a certain level of objectivity.

Second, preparing the experiments for each circuit type requires substantial work. We must perform a preliminary investigation of the circuit's sensitivity to heavy-ion radiation, and must design and implement the golden chip device. Therefore, the heavy-ion technique should be used primarily for complex VLSI circuits such as microprocessors, direct-memory access controllers, and the like. For other types of circuits, such as memory circuits and small- or medium-scale

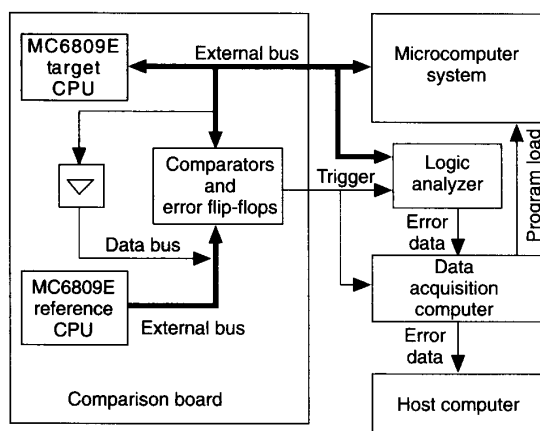


Figure 2. Block diagram of the experimental setup.

integration random logic, other fault injection techniques are sufficient and less costly. Given these limitations, the heavy-ion fault injection technique should be considered strictly as a complement to other fault injection techniques. Nevertheless, it is a very valuable technique, particularly for validating mechanisms designed to handle SEUs.

Experimental results

We designed our heavy-ion experiments with the Motorola MC6809E microprocessor to demonstrate the usefulness of the heavy-ion technique. In them, we investigated the error behavior of the microprocessor and evaluated several error detection mechanisms based on monitoring the behavior of the microprocessor. Our work also shows how simulation-based fault injection can help to interpret and understand results from physical fault injection experiments.

Experimental setup. Figure 2 shows a block diagram of the experimental setup for the MC6809 experiments. As the target system, we used a standard nonfault-tolerant, MC6809E-based microcomputer system. To detect errors immediately as they appeared on the output pins of the microprocessor, we used the golden chip method. To this end, we developed a comparison board containing both the irradiated CPU and the reference CPU. The board also contained comparators for comparing the output signals from the two CPUs and an array of error flip-flops for storing the state of all pins of both CPUs in the first erroneous bus cycle.

Our setup featured completely synchronized operation of the two CPUs and compared the output signals during each bus cycle. The irradiated CPU controlled the bus in the target system, while the reference CPU only listened to the bus to receive instructions and data during execution. Detection of a mismatch between the two CPUs caused the state of all

Table 2. First erroneous bus cycle (based on 1,000 errors).

Number of bits affected	Bus						
	A	D	C	A+D	A+C	D+C	A+D+C
>9	12	-	-	0	8	0	0
9	7	-	-	0	6	0	0
8	12	0	-	0	5	1	0
7	10	0	-	0	4	0	0
6	14	1	0	0	2	0	1
5	30	0	0	0	4	0	0
4	23	3	3	0	1	1	0
3	21	3	14	0	5	0	0
2	58	5	33	0	2	0	-
1	457	34	231	-	-	-	-
Total	644	46	270	0	37	2	1
Percent	64	5	27	0	4	<1	<1

pins on both CPUs to be stored in the error flip-flops. The mismatch also triggered a logic analyzer that recorded the bus signals of the irradiated CPU for 30 bus cycles before and 200 bus cycles after the error was detected. Thus, the error flip-flops identified in which signal or signals the error first manifested itself, while the logic analyzer data captured the error behavior of the irradiated CPU.

After detection of a mismatch, a data acquisition computer collected the error data from the error flip-flops and the logic analyzer. This computer then transferred the error data to a host computer used for storing and analyzing the error data. After this step, the data acquisition computer downloaded the software executed during the experiment to the target system and started a new fault injection experiment. The fault injection experiments proceeded under fully automatic control. Figure 1 showed a photograph of the target system, the comparison board, and the miniature vacuum chamber.

The program executed by the target system was an assembly language implementation of the Quicksort algorithm. It consisted of an infinite loop in which an array of pointers to 50 data records was first sorted according to the value of an integer variable in the data records and then reset to an unsorted state.

First erroneous bus cycle. We analyzed the data from the error flip-flops to determine the distribution of bit flips on the output pins in the first erroneous bus cycle. This data indicates where in the microprocessor the faults occurred. For example, errors first occurring in the control bus are likely to originate from the control section. Errors first occurring in the address or data bus could originate from either the register section or the control section, although the register section errors likely would dominate. Table 2 shows the

Table 3. Classification of errors.

Error class	No.	Percent
1. Control flow errors		
a. Errors that caused execution to diverge permanently from the correct program.	143	72* ± 6**
b. Errors that caused execution to diverge temporarily from the correct program	6	3 ± 3
c. Control flow errors that did not become effective within 200 bus cycles	4	2 ± 2
2. Data errors		
a. No addresses or control signals affected	9	5 ± 3
b. Addresses or control signals affected	30	15 ± 5
3. Other errors	8	4 ± 3
Total	200	
*Estimated probability that an error belongs to the given error class.		
**Corresponds to a 95-percent confidence interval.		

number of errors that affected the address, data, or control bus alone, and the number of errors that affected the various combinations of these buses. (The MC6809E has 16 address signals, eight data signals, and six control signals.) This data is based on 1,000 recorded errors. (In this table, columns A, D, and C list errors with bit flips only in the address, data, or control bus, respectively. Errors with bit flips in more than one bus are listed in the appropriate combined columns.)

Although we can reasonably assume that almost all errors originated from a single bit flip inside the chip, as much as 27 percent of the errors affected more than one bit in the first erroneous bus cycle. This result is not surprising, however, as the error propagation process would likely transform many single-bit errors to multiple-bit ones. For example, using the erroneous data in an arithmetic operation can transform a single-bit error in a data register into a multiple-bit error, while single-bit errors in the control section will very likely cause multiple-bit errors as they affect the function of the processor.

A majority of the errors affected either the address bus alone, 64 percent, or the control bus alone, 27 percent. Only 5 percent of the errors occurred in the data bus alone. About 4 percent of the errors manifested themselves both in the address bus and the control bus during the first erroneous bus cycle. Few or no errors first manifested themselves in other combinations of the buses. Eight errors caused the

processor to put the address bus, the data bus, and the read/write signal into a high impedance state. We can conclude from this data that the errors originated both from the register section and the control section of the microprocessor, and that the bit errors were reasonably well spread out within the chip.

Error behavior. We used an abstract error classification of the data recorded by the logic analyzer to describe the error behavior of the microprocessor. Two hundred of the recorded errors fell into one of three major groups: control flow, data, and other errors. We made this classification by manually inspecting disassembly listings of the error data recorded by the logic analyzer.

Control flow errors (class 1) are errors that caused the processor to diverge from the correct program—to read program words in incorrect sequence. This deviation was permanent (class 1.a) if the execution did not return to the correct track; that is, the error caused what is usually called a system program. The deviation could also be temporary (class 1.b). The third subclass (class 1.c) contains errors that had not caused the program to diverge from the correct program within 200 bus cycles, but that were predicted to do so later on.

Data errors (class 2) are errors that caused data to become erroneous without affecting the control flow. Errors that could not be detected by application-independent mechanisms—errors that did not result in the output of any incorrect address or control signals (class 2.a)—were separated from errors that application-independent mechanisms (class 2.b) could potentially detect.

Other errors (class 3) are those that did not affect either control flow or data. Thus, these errors did not result in program failure, although they did cause deviations from the specified behavior of the MC6809E. However, many of these errors could have caused program failure had the implementation details of the system been different. One such error is an error in the BA (bus available) status signal, which was not used in the target system. Table 3 shows the result of this classification.

Most errors, in all about 77 percent, affected the control flow in some way. Seventy-two percent of the errors caused a crash within the 200 bus cycles recorded by the logic analyzer. Two percent of the control flow errors did not become effective within the 200 bus cycles. Most of these errors changed the stack pointer so that an incorrect return from subroutine would later be made. With few exceptions, these errors would have caused the execution to diverge permanently from the correct program. Twenty percent of the errors caused data to become erroneous; that is, the program did not crash but the sort was not made correctly. Most of these data errors also affected addresses or control signals, while errors only affecting the data bus were rare.

Data errors and the duty-cycle effect. Perhaps the most pertinent question to ask when looking at the results of both

the error classification and the first erroneous bus cycle is: Why are the data errors so few? One explanation is that the MC6809E has only two 8-bit user registers for holding data (registers A and B), while there are five 16-bit user registers (including the stack pointers and the program counter) for holding address-related information. (Figure 3 shows the MC6809E user registers.) Thus, there are five times as many bits holding address-related information as there are bits holding data.

Another explanation is that errors more often become overwritten in data registers as their duty cycle normally is lower than for other registers. In this context, duty cycle means the proportion of time a register contains valid data—data that is going to be used later. A bit flip that occurs in nonvalid data has no effect and will eventually become overwritten when new valid data is loaded into the register. Registers with a low duty cycle therefore will have a high proportion of overwritten errors. We call this the duty-cycle effect.

Simulation experiments. We conducted a simulation-based fault injection experiment to study the duty-cycle effect in the MC6809E microprocessor. For this purpose, we designed a functional-level simulation model of the microprocessor and 64 Kbytes of memory using a hardware description language. The model only allowed injection of bit errors in the user registers, as these were the only hardware elements modeled. These experiments also used the Quicksort program as workload. They only injected single-bit errors from one to zero. As already mentioned, physical experiments showed that bit errors in the user registers occurred only in this direction. Randomly selecting a register bit and a bus cycle number in the execution of the Quicksort program defined the location and time for the injection of errors.

After injecting an error, our approach simulated the execution of the microprocessor for 200 bus cycles. It made a total of 2,690 error injection attempts. An error injection attempt succeeded if the correct value of the selected bit was one, so that an error could be injected by resetting the bit to zero. Consequently, an injection attempt failed if the correct value of the selected bit was zero. The experiment resulted in 1,042 (39 percent) successful error injections, which means that the register bits contained an average of 39 percent ones (61 percent zeros) during the execution of the Quicksort program.

Figure 4 shows the distribution of errors among registers, and for each register the proportion of effective, overwritten, and latent errors. Effective errors propagated to the output pins. Latent errors did not propagate to the output pins, nor were they overwritten before the simulation terminated after 200 bus cycles. The diagram clearly shows a high percentage of overwritten errors for the data registers—registers A and B—while the percentage of overwritten errors was very low in the stack pointer (S register) (2 percent) and the program counter (PC register) (7 percent). As many as 85 percent of the effective errors originated from either the pro-

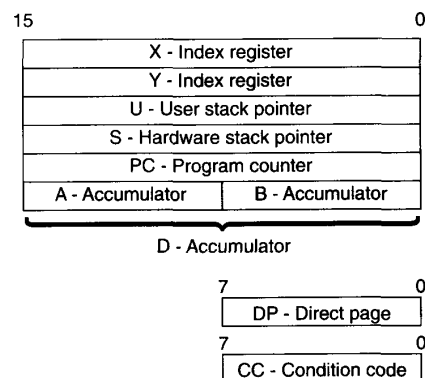


Figure 3. MC6809 user registers.

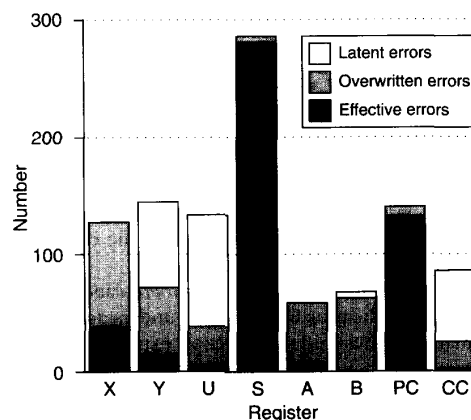


Figure 4. Error distribution from simulation-based fault injection (based on 1,042 errors).

gram counter or the stack pointer; only 2.5 percent of the effective errors originated from the A and B registers. These findings clearly demonstrate the impact of the duty-cycle effect. Note especially that this effect creates a workload dependency for the error behavior of microprocessor, as the duty cycle of the registers varies for different programs.

Note also that the number of successful error injection attempts is lower for the A, B, and CC registers, as these are 8-bit registers, while the others are 16-bit registers. The total number of successful error injection attempts in each register depends on the distribution of zeros and ones in the register during the execution of the Quicksort program. For example, the average number of ones was higher for the

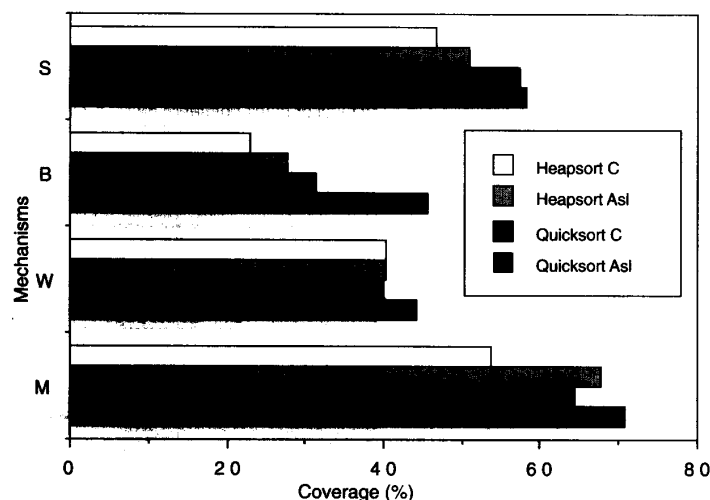


Figure 5. Detection coverage for watchdog error detection mechanisms (based on 600 errors for each program).

stack pointer than for other registers, therefore a larger number of errors was injected into the stack pointer. The asymmetric sensitivity of the memory elements also leads to a workload dependency, as the distribution of effective errors depends on the locations of the stack area, the data area, and the program area used by the workload.

These results clearly demonstrate how simulation-based fault injection can help us interpret and understand observations made in physical fault injection experiments. The results also show the importance of using a realistic workload when validating a fault-tolerant system by fault injection.

Note, however, that a comparison of the results from the physical and the simulated experiments showed that we cannot use this type of high-level simulation to accurately model the effects of heavy-ion-induced faults. The error behavior of the microprocessor was much more complex in the physical experiments. For example, the percentage of multiple-bit errors observed on the output pins in the first erroneous bus cycle was 31 percent in the physical experiments, but only 6 percent in the simulations. The error propagation must therefore be more complex in the real circuit. This is an interesting observation as the simulation model basically has the same capacity to inject faults as software implemented fault injection—both are capable of injecting faults in the user registers of the processor. Consequently, we cannot use software-implemented fault injection to accurately emulate the effects of heavy-ion induced faults. (Remember that we use the term software-implemented fault injection for software-based fault injection in a physical system.)

Error detection mechanisms. We have used heavy-ion

based fault injection to evaluate several simple error detection mechanisms for the MC6809E. These error detection mechanisms are intended for implementation by means of a watchdog processor. A watchdog processor is a small coprocessor that monitors various properties of the behavior of the CPU. Examples of such properties are memory access behavior, control flow, and opcode fetching. Using a watchdog processor allows detection of CPU errors at a lower cost than using duplicated CPUs and comparing the output signals. However, the error coverage for a watchdog processor is not as high as for duplicated CPUs. The coverage also depends on the erroneous behavior of the processor, which makes it particularly important to experimentally evaluate the coverage.

Now, we will present the results of an evaluation of four such error detection mechanisms. Two of them, called mechanisms S and B, monitor the control flow,

while the other two, mechanisms M and W, monitor the address access behavior. As workloads in this evaluation, we used four programs that perform the same sorting task but are implemented differently. The programs used either the Quicksort algorithm or the Heapsort algorithm. We implemented each algorithm in one assembly language version and one C-language version. The Quicksort assembly language program was identical to the program used for the error classification and simulation experiments already presented. Using these four programs enabled us to study how much the error coverage varied for similar programs. Let us first describe how the error detection mechanisms work.

Mechanisms S and B. We can consider a computer program as a number of blocks, each consisting of a sequence of instructions ended by a branch instruction. Within each program block, instructions execute in sequence until the branch instruction is reached. After the branch instruction, execution continues within another block, or loops back within the same block. Mechanism S monitors the program flow within each block, while mechanism B monitors the branches between blocks. Thus, mechanisms S and B complement each other.

Mechanism S uses the opcode of the current instruction to obtain the number of words occupied by this instruction in program memory, that is, the instruction size. It calculates the address of the next instruction by adding the instruction size to the address of the current instruction. The assertion of the last instruction cycle (LIC) signal by the MC6809E precedes the start of each instruction. When the CPU fetches the next instruction (indicated by the preceding assertion of

LIC), the mechanism compares the address generated by the processor to the computed address. Any discrepancy indicates an error. Illegal opcode detection is inherent in this mechanism, as each opcode value is mapped to an instruction size. When no new instruction has been fetched within the execution time of the most time-consuming instruction (20 cycles for MC6809E), that also indicates an error.

Unlike other proposed techniques for concurrent execution flow checking, this mechanism requires no information about the program executed. Nor do any signatures need to be inserted in the instruction stream. However, the mechanism must be able to derive the instruction size from each opcode and to recognize the first cycle of each instruction. The next address after a branch instruction cannot be derived from the opcode if the branch is taken. We should therefore use mechanism S in conjunction with mechanism B.

Mechanism B checks that branches are made to valid destination addresses. This check occurs during the opcode fetch of instructions succeeding branch instructions. Before the checking can begin, the detection mechanism must know the permitted branch destination addresses. A tag-bit in memory that marks valid destination addresses could provide this information. The next opcode address in sequence after a conditional branch must also be marked as a valid destination address because the mechanism cannot decide whether the branch will be taken. However, we can avoid using a tag-bit for the next opcode address if we combine mechanism B with mechanism S. This reduces the total number of tag-bits and thereby the risk that an incorrect branch will be taken to a destination address that is valid for another branch instruction, in which case the illegal branch will not be detected.

Mechanism M. This mechanism detects any access to memory outside a permitted memory area. MC6809E has a 16-bit address bus and thus a 64-Kbyte address space. In our experiments, we designated the addresses C000-CFFF (4 Kbytes) as program area and D000-DFFF (4 Kbytes) as data area. Any access outside the program and data areas causes mechanism M to signal an error.

Mechanism W. This mechanism detects any attempt to write into memory locations not designated as alterable. Any write access outside of the 4-Kbyte data area causes mechanism W to signal an error.

The experimental setup we used to evaluate the error detection mechanisms was the same as the one previously described. However, the logic analyzer recorded only 85 bus cycles after the golden chip device detected an error. A total of 2,400 errors, 600 for each sorting program, was recorded. We did not physically implement the error detection mechanisms. Instead, we implemented a program that determined the coverage of the mechanisms by an off-line analysis of the data collected from the logic analyzer.

Figure 5 shows the error detection coverage for the individual mechanisms. The diagram shows that mechanism M

Table 4. Combinations of detection mechanisms.

Combinations of mechanisms	All errors (based on 2,400 errors)		Program crashes (based on 210 errors)	
	Coverage (%)	Median latency	Coverage (%)	Median latency
S,B,M,W	79	1	99	2
S,M,W	77	1	97	2
S,B,M	77	1	99	2
S,B	63	2	99	3
S, M	75	1	97	2
B,M	69	4	93	5

had the highest coverage, followed by mechanisms S, W, and B. The results clearly show the variation in coverage for the different programs. The coverage figures vary from 23 percent for mechanism B for the Heapsort C program to 71 percent for mechanism M for the Quicksort Assembly program. Overall, the coverage figures are not very impressive for the individual mechanisms. However, we can obtain higher coverage by combining several mechanisms.

Table 4 shows the coverage obtained for various combinations of the error detection mechanisms. It shows the coverage and the median error detection latency for all 2,400 errors. Here, detection latency means the time between an error first appearing on the bus until the watchdog mechanism detecting it. The latency is measured in number of bus cycles. (One bus cycle equals 1 μ s, as the clock frequency of the processor was 1 MHz.) We obtained a coverage of 79 percent with a median latency of 1 bus cycle when all four mechanisms were combined.

Table 4 also shows the coverage and latency for 210 control flow errors of class 1.a (permanent control flow errors). These errors came from a classification of 360 errors (90 for each workload). The results show that the coverage was 99 percent for this error class when all four mechanisms were combined. In this case, the median latency was two bus cycles. We also obtained a coverage of 99 percent when combining only the two control flow checking mechanisms, S and B, but the median latency then was 3 bus cycles. Further analysis showed that we could detect about 50 percent of the data errors that affected the address signals (class 2.b) and, as expected, none of the pure data errors (class 2.a).

THIS WORK SHOWS THAT THE WATCHDOG mechanisms provide good coverage and short detection latency for control flow errors. However, they should be complemented with other mechanisms, for example, executable asser-

tions or reasonable checks, to provide better coverage for data errors. We provide a more detailed presentation of the results from the evaluation of the watchdog error detection mechanisms elsewhere.¹¹

Our work shows how heavy-ion radiation can be used for injecting faults internally in integrated circuits to validate fault-handling mechanisms and to study the effects of transient faults in computer systems. The ability to inject faults at internal locations in VLSI circuits is becoming increasingly important with the proliferation of safety-critical and fault-tolerant systems using VLSI technology. However, we must view the heavy-ion technique strictly as a complement to other fault injection methods, as it does not control the time and location of the injection of faults.

More research is needed to solve the practical limitations of the heavy-ion technique. One problem is that the method requires special facilities for safely handling the Cf-252 source. Although the use of the miniature vacuum chamber minimizes the safety hazards during the actual fault injection experiments, the method requires access to a vented hood in a radiologically controlled area for opening the chamber to exchange the irradiated circuit. Opening circuit packages and cooling circuits in the miniature vacuum chamber presents other potential problems. One solution would be to develop a method for depositing Cf-252 on top of the chip surface. This way, we could enclose the radioactive material in the circuit package and neither vacuum nor special precautions for radiation safety would be required. We could even confine the radioactive material to specific areas of a VLSI chip, thus achieving a specific error behavior.

However, before developing such a method we must fully assess the usefulness of the heavy-ion technique for validating fault-handling mechanisms in dependable computing systems. Therefore, we are now planning fault-injection experiments that will compare the heavy-ion technique with several other fault-injection techniques. ■

Acknowledgments

This work was funded by the Swedish National Board for Industrial and Technical Development (NUTEK) under contract numbers 86-3585 and 9302642. We thank Jan Torin, who first suggested that heavy-ion radiation could be used for fault injection, for the many suggestions, encouragement, and support he has provided. We also thank Eskil Johnson for providing insight into circuit-level simulations, and Hans Bergstrand, Lennart Hansson, and Ghassem Miremadi for their contributions to the development of the experimental setup.

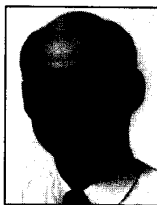
References

1. J. C. Laprie, "Dependable Computing and Fault-Tolerance: Concepts and Terminology," *Proc. 15th Int'l Symp. Fault-Tolerant Computing*, IEEE Computer Society Press, Los Alamitos, Calif., 1985, pp. 2-11.
2. J. Arlat et al., "Fault Injection for Dependability Validation: A Methodology and Some Applications," *IEEE Trans. Software Engineering*, Vol. 16, No. 2, CS Press, Feb. 1990, pp. 166-182.
3. J.H. Lala, "Fault Detection, Isolation, and Reconfiguration in FTMP: Methods and Experimental Results," *Fifth AIAA/IEEE Digital Avionics Sys. Conf.*, Nov. 1983, pp. 21.31.1-21.3.9.
4. A.C. Merenda and E. Merenda, "Recovery/Serviceability System Test: An Improved Testing Approach," *Proc. 22nd Int'l Symp. Fault-Tolerant Computing*, CS Press, July 1992.
5. J.H. Barton et al., "Fault Injection Experiments Using FIAT," *IEEE Trans. Computers*, Vol. 39, No. 4, CS Press, Apr. 1990, pp. 575-582.
6. G.A. Kanawati, N.A. Kanawati, and J.A. Abraham, "FERRARI: A Tool for the Validation of System Dependability Properties," *Proc. 22nd Int'l Symp. Fault-Tolerant Computing*, CS Press, July 1992.
7. J. Sosnowski, "Experiments with Transient Fault Upsets in Microprocessor Controllers," *Proc. Third GII/TG/GMA Int'l Conf. Fault-Tolerant Computing Sys.*, Springer-Verlag, Berlin, Heidelberg, 1987, Sept. 1987, pp. 153-164.
8. J. Karlsson et al., "Two Fault Injection Techniques for Test of Fault Handling Mechanism," *Proc. Int'l Test Conf.*, Piscataway, N.J., Oct. 1991, pp. 140-149.
9. J. H. Stephen et al., "Cosmic Ray Simulation Experiments for the Study of Single Event Upsets and Latch-up in CMOS Memories," *IEEE Trans. Nuclear Science*, Vol. NS-30, No. 6, IEEE, Dec. 1983, pp. 4464-4469.
10. D. Mapper et al., "An Experimental Study of the Effect of Absorbers on the LET of Fission Particles Emitted by Cf-252," *IEEE Trans. Nuclear Science*, Vol. NS-32, No. 6, IEEE, Piscataway, N.J., Dec. 1985, pp. 4276-4281.
11. U. Gunneflo, J. Karlsson, and J. Torin, "Evaluation of Error Detection Schemes Using Fault Injection by Heavy-Ion Radiation," *Proc. 19th Int'l Symp. Fault-Tolerant Computing*, CS Press, June 1989, pp. 340-347.



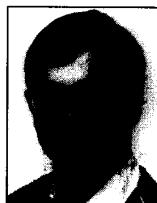
Johan Karlsson is an assistant professor in the Department of Computer Engineering at Chalmers University of Technology. His research interests focus on experimental validation of dependable computing systems by means of physical and simulation-based fault injection.

Karlsson received an MS in electrical engineering and a PhD in computer engineering from Chalmers. He is a member of the IEEE and the Computer Society.



Peter Lidén is a PhD research student in the Department of Computer Engineering at Chalmers University of Technology. His research interests include physical and simulation-based fault injection, fault modeling, and self-checking circuits.

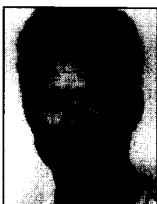
Lidén received an MS degree and a licentiate of engineering degree in computer engineering from Chalmers. He is a 1989 recipient of the John Ericsson medal and a student member of the IEEE.



Ulf Gunneflo recently joined Carlstedt Elektronik AB, where he is currently working on research and development. Earlier, he served as an assistant professor in the Department of Computer Engineering at Chalmers University of Technology. His research interests include fault-tolerant computing and parallel processing.

Gunneflo received an MS in electrical engineering and a PhD in computer engineering from Chalmers.

Direct questions concerning this article to Johan Karlsson, Department of Computer Engineering, Chalmers University of Technology, S-412 96 Göteborg, Sweden; johan@ce.chalmers.se.



Peter Dahlgren is a PhD research student in the Department of Computer Engineering at Chalmers University of Technology. His research interests include simulation-based fault injection, switch-level fault modeling, and self-checking circuits.

Dahlgren received an MS degree and a licentiate of engineering degree in computer engineering from Chalmers. He is a 1989 recipient of the John Ericsson medal and a student member of the IEEE.



Rolf Johansson is a PhD candidate in the Department of Computer Engineering at Chalmers University of Technology. He also works for Saab Scania Combitech and for Saab Ericsson Space. His research interests include physical and simulation-based fault injection, self-checking logic, and error-correcting codes.

Johansson received an MS degree in engineering physics from the University of Lund and a licentiate of engineering degree in computer engineering from Chalmers.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card

Low 153

Medium 154

High 155

Read IEEE Micro

in April for articles from
the IEEE Computer
Society's
Hot Chips V Symposium

Hot Chips!

- Motorola's PowerPC Architecture and Implementation
- Silicon Graphics TPE Microprocessor Design
- Hewlett-Packard's VLSI Chipset for Low-Cost Graphics and Multimedia Workstations
- DEC's Architecture, Packaging, and Testing of BIPS