Chapter 2
Background

This chapter introduces the main technical terms used in this text, describes the microprocessor architecture used as a case study, and discusses background information required for better enlightenment of the topics in this book.

2.1 Basic Concepts of Dependable and Secure Computing

This section presents the basic set of definitions that will be used throughout this work. The definitions encompass from defects and upsets that occur in individual logic gates to fault, error, and failure.

2.1.1 Defect, Upset and Fault Definitions

Defect or upset is defined as unintended differences between the implemented system and its intended function. It can be commonly a manufacture defect, for example, or transient upsets that happen during some perturbation of the environment.

Fault is then defined as a logic level abstraction of a physical defect or upset. It is used to describe the change in the logic function of a device caused by a defect or upset. Fault can be described as a deviation from the expected behavior of the logic. Faults can be transient, intermittent or permanent. Transient faults occur and then disappear. They are transient effects that may occur during the lifetime of the component and it exists for a short period of time. Intermittent faults are characterized by a fault occurring, then vanishing, then reoccurring, and so on. An example of intermittent faults is signal interference, such as the cross-section between connection lines. Permanent faults continue to exist in the system until the faulty component is repaired or replaced. They usually happen due to manufacturing problems. Some defects or upsets may be masked by the electrical properties of the device and no fault may be observed. Whenever there is a fault in the circuit, there is also the possibility of an error in the near future. However, there are a few occasions where faults can be masked by the logic, by electric characteristics, or by the application

J. R. Azambuja et al., *Hybrid Fault Tolerance Techniques to Detect Transient Faults in Embedded Processors,* DOI 10.1007/978-3-319-06340-9_2,

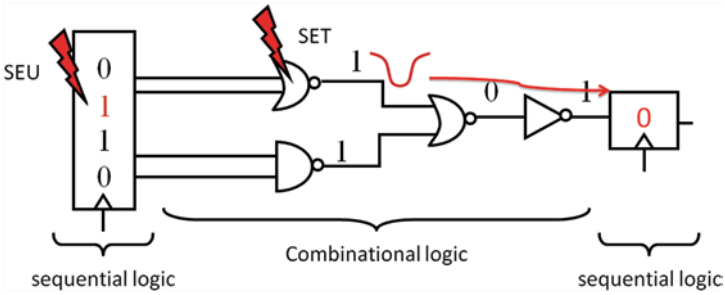


Fig. 2.1 SEU and SET effects on a circuit

running on the circuit. In such cases, the fault does not lead to an observed error. Error is considered to be a wrong output value produced by a defective system.

With nanometer dimension technologies, transistors have become more susceptible to faults caused by radiation interference. It happens due to reduced threshold voltages, reduced node capacitances, and tightened noise margins (BAUMANN 2001). Such faults can be caused by energized particles present in space or secondary particles such as alpha particles, generated by the interaction of neutron and materials at ground level (INTERNATIONAL 2005). Integrated circuits operating in a space environment are sensitive to these particles and can be affected mainly by transient ionization and long term ionizing damage.

In the following, we will discuss the main effects from radiation interference that cause upsets in integrated circuits.

2.1.1.1 Single Event Effect (SEE)

Transient ionization may occur when a single radiation ionizing particle strikes the silicon, creating a transient voltage pulse, or a Single Event Effect (SEE). This effect can be destructive or non-destructive. An example of destructive effect is Single Event Latch up (SEL) that results in a high operating current, above device specifications, that must be corrected by a power reset. Non-destructive effects, also known as soft errors, are transient effects provoked by the interaction of a single energized particle in the PN junction of an off-state transistor (DODD 2004). When the transient pulse occurs in a memory element, such as a flip-flop, it is classified as Single Event Upset (SEU). When the particle hits a combinational element, such as a multiplexer element, and thus inducing a pulse in the combinational logic, the upset is classified as Single Event Transient (SET).

Figure 2.1 shows examples of SEU and SET effects in a circuit. On the left, one can see the SEU effect. A particle, represented by the bolt, hits the sequential logic (which could be seen as a register), changing the stored value from “0010” to “0110”. This effect directly affects the rest of the circuit, changing the value stored in the sequential logic on the right from “1” to “0”. In the middle, one can see a particle hitting the NOR gate and causing a voltage pulse in the combinational logic.

When propagated, the pulse hits the sequential latch window on the sequential logic to the right, which registers the incorrect value "0", instead of a "1". Such effects may be masked by the circuit, as discussed in the following subsections.

Soft errors can be detected and corrected by the system's logic, meaning that it does not require a hard reset to recover from an error. Sections 7.1.2 and 7.2.2 present neutron irradiation experiments simulating the effect of SEE in Flash-base and SRAM-based FPGAs, while Chaps. 5 and 6 present fault injection simulation experiments simulating SEEs at RTL level and in the configuration memory bitstream, respectively. In this work, SEUs and SETs will be used to describe transient faults that the proposed techniques can cope with.

2.1.1.2 Total Ionizing Dose (TID)

The long term ionizing damage is also known as Total Ionizing Dose (TID). It is caused by the interaction of energized particles with atoms of the silicon. Photon-induced damage is initiated when Electron-Hole-Pairs (EHP) are generated along the track of secondary electrons emitted via photon-material interactions. EHPs are created from a fraction of the kinetic energy of the incident particles. Some of them are annihilated due to recombination, but a few remain in the silicon. The remaining EHPs may fall into deep traps in the oxide bulk or near the Si/SiO_2 interface, forming trapped positive charges (BARNABY 2006; OLDHAM 2003). By doing so, TID can affect the system by shifting the threshold voltage, generating leakage current and timing skews and even leading to functional failures. Sections 6.1.2 and 6.2.2 present neutron irradiation experiments using Flash- and SRAM-based FPGAs, respectively, while Sect. 6.1.1 discusses the effects of TID in SRAM-based FPGAs.

In this book, we will refer fault as the SET pulse that may occur in the combinational logic and as the SEU that is the bit-flip that may occur in the memory element.

2.1.2 Error and Failure Definitions

The design of fault tolerant systems consists in preventing a fault to cause an error and consequently a failure in the implemented system. Therefore, there is a cause-effect relationship from the particle hit (fault) to the erroneous result (system failure), as demonstrated in Fig. 2.2. In this work, we will use the definition presented in Avizienis (2004).

In order to define error and failure, we first have to define a system. A system is an entity that interacts with other entities, such as other systems, hardware, software, and the physical world. A system follows a functional specification, composed of several different functions. The behavior of a system is what it does to implement its functions and is described by a sequence of states. Finally, the service delivered by a given system is its behavior, as it is perceived by its users.

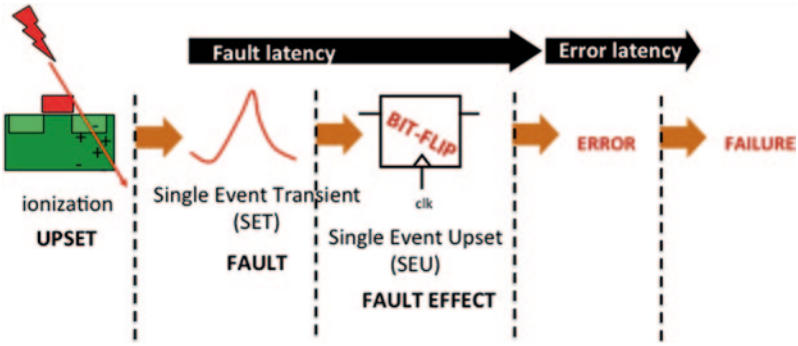


Fig. 2.2 Upset, fault, error and failure chain-effect for SET and SEU

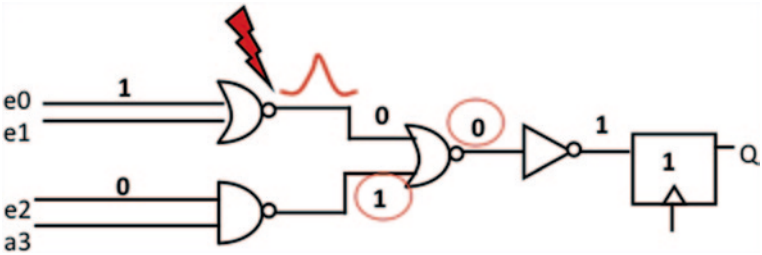


Fig. 2.3 Logical masking

Failure is the abbreviation of service failure and is defined as a system malfunction, or in other words, when the delivered service deviates from the correct one. The delivered service is considered correct when it is according to the system specification. When the service specification includes a set of several functions, the failure of one or more of the services implementing the functions may lead the system to a degraded mode that still offers a subset of needed services. We define this case as a partial failure.

Error is defined as the deviation in one of the system's sequence of states. Such deviation may compromise a system service, thus leading to a service failure. It is important to note that an error not always leads to a failure.

By defining fault, error and failure, one can notice that a failure can always be seen by the user, since it leads to a system malfunction. Faults can be latent in the circuit until manifested as an error. There are detection techniques that can detect faults and there are techniques that can detect errors.

Faults can also be masked by three different ways in the circuit: logical masking, electrical masking and latch window masking. Figures 2.3, 2.4 and 2.5 show each of them, respectively.

Logical masking is when the logic of the gate being hit by a fault masks its effect. The basic logic operators AND and OR both have a dominant value as property,

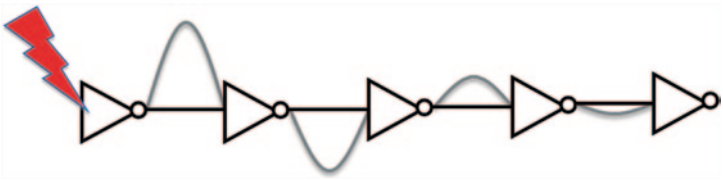
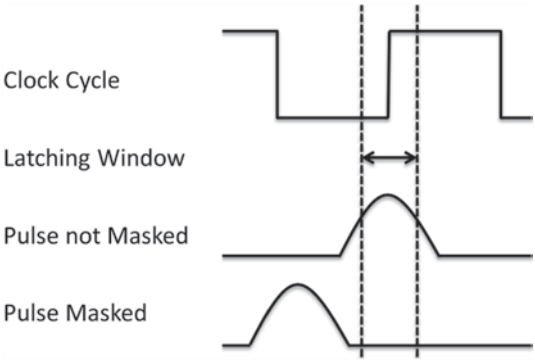


Fig. 2.4 Electrical masking

Fig. 2.5 Latch window masking



which forces the output to a result based only on a single input. The same property applies to their NOT forms, NAND and NOR. The OR operator, for example, has the dominant value of '1'. Whenever one of its inputs is '1', the output will also be '1', independently of the other inputs. When negating the output (NOR), the dominant value remains the same, only inverting the output of the gate. Figure 2.3 shows a NOR gate being hit by a particle and forcing its output to '1'. As one can see, the fault does not propagate through the circuit, as the following NOR gate has a second input with its dominant value, '1', and thus forcing the output of that gate to '0'. In other words, the fault has no effect in the circuit due to its logic properties. Considering that any circuit can be implemented using AND and OR operators, we can extend this property to more complex ports and circuits.

Electrical masking happens when the propagation of the pulse is weakened by the logic, such as the one shown in Fig. 2.4. When a particle hits a NOT gate, it generates a high voltage pulse that affects the following gates. When propagating through the circuit, the pulse is weakened until masked. This type of masking has become more relevant in the past years due to reduced threshold voltages. In order to reduce the power consumption of circuits with increased number of transistors, newer semiconductor fabrication technologies have been reducing threshold voltages and therefore increasing the circuit path necessary to electrically mask a voltage pulse.

Latch window masking is related to the memory logic. Flip-flops are the main elements to store data within a circuit (besides actual memory), and are commonly

implemented by using two latches with opposite latching voltages. Because of this design, flip-flops have a setup time and a hold time, which are called latch window when summed, and represent the time when the data must be stable in the input of the flip-flop in order to be correctly stored in its logic. This property means that any value in the flip-flop's inputs outside the latching window will be discarded by the memory logic. Figure 2.5 shows a clock cycle and a register's latching window. When the fault-induced voltage pulse starts before the latching window and lasts until after the windows closes, the fault is stored in the logic. When the fault-induced voltage pulse does not last until the latch window closes (as shown in the bottom), the fault is masked. Newer semiconductor fabrication technologies have been increasing operating clock frequencies and therefore reducing the clock period. By doing so, the latch window masking is being also reduced and SET are becoming even more common than in the past years.

2.2 MIPS Architecture

The case study processor used in this work is based in the Microprocessor without Interlocked Pipeline Stages (MIPS) architecture. It has a standard processor architecture based on the Reduced Instruction Set Computing (RISC) instruction set. The basic idea behind RISC is to use simple instructions, which enable easier pipelining and larger caches, while increasing its performance. The MIPS architecture can be seen since 1985 in commercial applications, from workstations to Windows CE devices, routers, gateways and PlayStation gaming devices.

Among the different MIPS architecture processors, there is the miniMIPS (HANGOUT 2013), which will be used in the work as case-study microprocessor. The miniMIPS is an open source processor that has a reduced instruction set from the original MIPS architecture, with 52 instructions. It is described in hardware description language Very-high-speed integrated circuits Hardware Description Language (VHDL) and therefore can be logically simulated and synthesized into programmable circuits, such as FPGAs. The miniMIPS can be implemented using the Harvard memory model, where the program and data memory are separated physically in two different memories, or the Von Neumann, where program and data memory share the same physical memory. It has five stages pipeline: instruction address calculation (PC), instruction fetch (FETCH), instruction decode (DECODE), execution (EXEC), and memory access (MEM). A model of the miniMIPS pipeline is shown in Fig. 2.6.

The miniMIPS microprocessor has thirty-two 32-bit registers in its register bank. It also has a PC with a simplistic logic, since it has fixed size instructions and static branch prediction. Besides the PC, the microprocessor has other special purpose registers, such as the Stack Pointer (SP), Global Pointer (GP), Frame Pointer (FP), Return Register (RA), and Zero (always has the value 0), which can all be found in the register bank. The miniMIPS uses a gcc cross-compiler to translate C code into executable code.

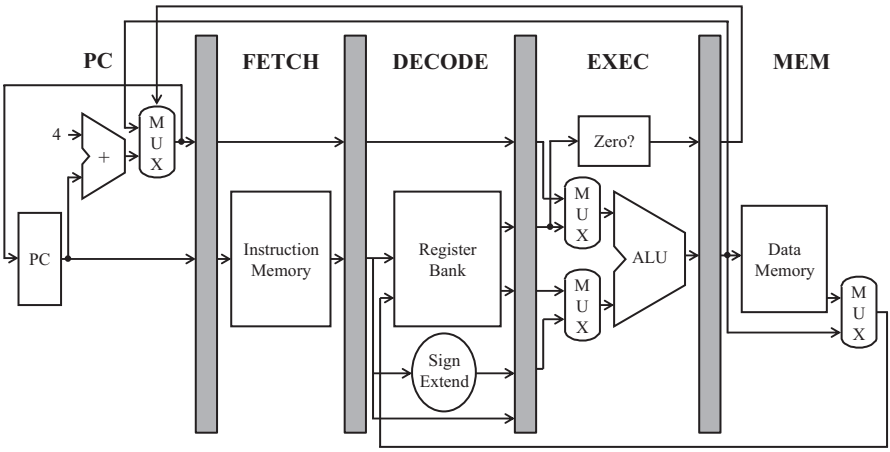


Fig. 2.6 Pipeline architecture of the miniMIPS

Table 2.1 MIPS' instruction format

Type	-31- format (bits) -0-					
R	opcode (6)	rs (5)	rt (5)	rd (5)	shamt (5)	funct (6)
I	opcode (6)	rs (5)	rt (5)	immediate (16)		
J	opcode (6)	address (26)				

The instruction set used by miniMIPS has a fixed size of 32 bits, from which only two have access to the memory: the load instruction and the write instruction. Instructions have a 6-bit opcode and are divided in three classes: type-R, which specify three registers (*rs*, *rt* and *rd*), plus a shift value (*shamt*), and a function field (*funct*); type-I, which specify two registers (*rs* and *rt*) and a 16-bit immediate value; and type-J, which do not specify any register, only a 26-bit address. Figure 2.7 shows these classes in detail (Table 2.1).

The miniMIPS microprocessor has been chosen as a case-study to this work due to five main reasons: (1) it was presented in 1985 and is still used in the industry, (2) it is largely used in the literature, (3) it implements a simple but efficient RISC architecture with a 5-stage pipeline, which follows modern processor models, such as Intel's and ARM's, (4) it has a stable open version since 2009, (5) it has been simulated and implemented in various platforms, from FPGAs (both SRAM-based and flash-based) to ASICs. The miniMIPS version used in this work was initially developed by the École Nationale Supérieure d'Electronique et de Radioélectricité de Grenoble (ENSERG), made open source in Hangout (2013), and slightly improved at UFRGS (branch prediction module and memory controllers were added).

In this book, we will use a VHDL model of the miniMIPS that can be logically simulated and synthesized to ASIC, or programmed to FPGA platforms.

2.3 SEE in MIPS Processors

A processor is nothing more than a group of sequential and combinational circuits combined in one component. This combination of different circuits induces processors to be sensitive to different radiation effects in different areas. A processor could be roughly divided into five logical groups according to its area (program memory, data memory, register bank, control path, and data path) and into two logical groups when relating to the effect of a fault (data flow and control flow). In the following subsections, we will address how SEEs affect each part of a processor and their effects.

2.3.1 SEEs Divided by Sensitive Areas of a Processor

Memories are sequential circuits and therefore very sensitive to SEUs. Due to their regular physical structure, they are optimized to fit in smaller die areas than normal circuits and normally with higher operating clock frequencies. That means that radiation effects, such as multiple-bit upset due to a single particle, are intensified in memory components. There are two main types of available memory: flash and SRAM memory. Flash memories are less sensitive to radiation effects than SRAM memories, because they require voltages higher than normal operating voltages to be written (change their current state), typically higher than 5 V. Therefore, a particle must have more energy in order to cause an upset in a flash memory cell. On the other hand, such memories have as drawbacks the fact that it has a finite number of write-erase cycles, meaning that a memory cell can only be written around 100,000 times before deteriorating its integrity. Also, flash memories normally include a circuit to pump up the voltage (for writing purposes), and this circuit is sensitive to radiation effects. The SRAM memory is more sensitive to radiation effects because it operates in normal voltages, but has better performance and power consumption. Also, SRAM memories do not have the finite number of write-erase cycles.

The memory organization of a processor can be with program and data memory combined in the same physical memory (Von Neumann), or separated (Harvard). When separated, the program memory is usually stored in a flash memory and the data memory in an SRAM memory. By doing so, it is possible to reduce the number of upsets in the program memory, while the data memory can be protected by fault tolerance techniques. When sharing the same memory, program and data memories are typically implemented on a SRAM memory. Because of the fact that fault tolerance techniques are too expensive to protect the program memory, low-level approaches, such as Error Detection And Correction (EDAC) must be implemented on the memory.

The register bank is mostly a sequential circuit, just like the program and data memory. Because of that, it is very sensitive to SEUs. The register bank can be implemented over a SRAM memory or by using flip-flops. In the first case, the same principles from the data program are applied. In the second case, hardware

replication can be used, or even software-based technique to replicate the information stored in the registers. The miniMIPS has thirty-two 32-bit registers, resulting in a total of 1024 bits, which is a big number, when considering radiation effects. As a comparison, the register bank represents almost half of signals describing the miniMIPS.

The data path represents the computing circuit of the processor. It is defined as the circuit leading from a stored value (in the memory or in the register bank), through the Arithmetic and Logic Unit (ALU), and back to a store element. It is composed of both combinational and sequential logic, since the data path not only processes data, but also crosses the register barriers from the pipeline stages. Because of that, it is sensitive to SEUs (in the pipeline registers) and SETs (in the computing logic, such as the ALU). The effect of a fault in the data path usually leads to an erroneous result in the end of the computation, but hardly leads to an infinite loop, or a control flow error.

The control path is defined as the decision logic of a processor. It is responsible for calculating the next instruction to be fetched and setting the internal flags, such as to command the ALU to sum or subtract, and a branch to be taken or not. The control path is mostly combinational, but since it has to cross the pipeline stages, also has sequential logic. The main difference between the control path and the data path is that an error in the control path will most likely lead to control flow errors, such as a branch being taken, when it should not have. Such control flow errors may cause an erroneous result in the end of the computation or even an infinite loop.

Figure 2.7 shows a detailed view of the architecture of the miniMIPS with the sensitive areas.

2.3.2 SEEs Divided by Effect on a Processor

Faults can be classified as having data or control flow effect in a processor. Data flow effect is defined as an error in a variable during the computation. It means that the program was correctly executed, but with an erroneous result. An example would be the instruction "Registers A = Register B + Register C", where the value stored in A would be "Register B + (Register C + 1)", due to an SEU that happened in Register C. It means that the processor correctly performed the sum in the ALU, but register C had an incorrect value. Control flow effect is defined as an error in the program execution. It means that the variables were correct, but the computation was incorrect. An example would be the same instruction, "Registers A = Register B + Register C", where the value store in A would be "Register B – Register C", due to an SET in the ALU that subtracted the registers, instead of summing them.

In order to differentiate a control flow from a data flow error, we check the PC evolution and compare it with a golden module. In case of a mismatch, the fault is classified as a control flow effect. If not, it is classified as a data flow effect. In some cases, a fault with a data flow effect may cause a control flow effect. An example could be an error in a register used to decide whether a branch should be taken or not. In such cases, we consider it as a control flow effect.

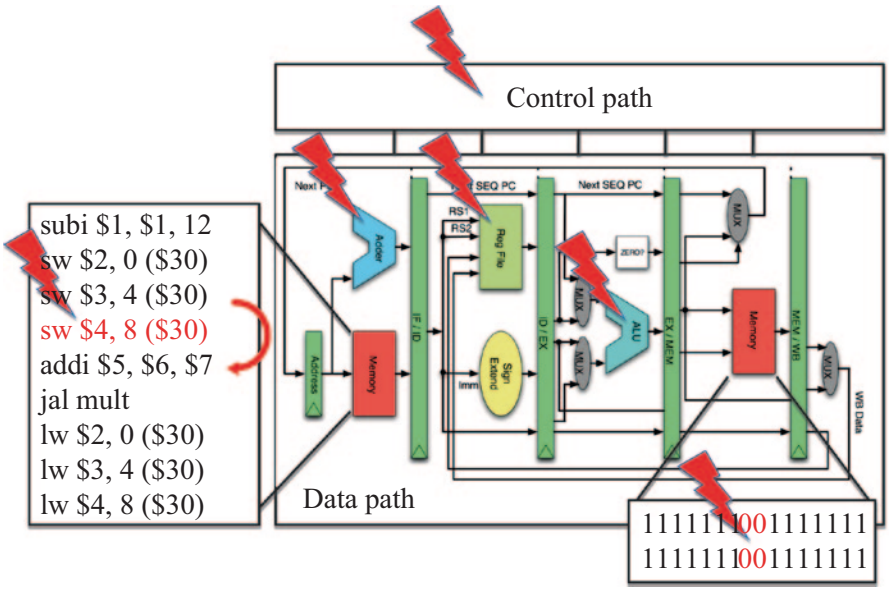


Fig. 2.7 miniMIPS sensitive areas under SEE

The relation between the location of a fault and its effect is not direct. A fault in the register bank not necessarily will have a data flow effect on a processor. Likewise, a fault in the control path will have a control flow effect.

Figure 2.7 shows the difference between a control flow and a data flow effect in a processor. On the left, one can see a control flow effect, where a jump in the program execution causes an error. On the bottom right, one can see the data memory with errors due to a data flow effect fault.

2.4 Fault Injection and Testing

Although the effect of faults is increasing, the rate is not yet sufficient to test fault tolerant techniques at ground level. In order to do so, fault emulation and testing is necessary. In this Section, we will go over a few options to do so, such a software fault injection by simulation, fault injection in the FPGA's memory configuration bitstream and irradiation experiments.

2.4.1 Fault Injection by Simulation

Fault injection by simulation can be done by injecting faults at logical or electrical levels in commercial simulators, such as ModelSim from Mentor or iSim

http://www.springer.com/978-3-319-06339-3

Hybrid Fault Tolerance Techniques to Detect Transient Faults in Embedded Processors

Azambuja, J.R.; Kastensmidt, F.; Becker, J.

2014, XVIII, 94 p. 37 illus., 11 illus. in color., Hardcover
ISBN: 978-3-319-06339-3