Integration of Soft Errors in Functional Safety: A Conceptual Study

Jens Vankeirsbilck, Hans Hallez and Jeroen Boydens

Abstract - **Embedded systems are used for both safety and non-safety critical applications. Safety critical systems must have a very low failure rate, as a failure can cause injury or even death. This paper presents soft errors as a cause for failure. Soft errors are systematic failures that lead to bit-flips in registers or other memory regions and affect the software in execution. The soft error and its detection techniques are linked to requirements imposed by the functional safety standards.**

Keywords – **Functional Safety, Soft Errors, Embedded Systems**

1. INTRODUCTION

Embedded systems are used more and more for different types of applications, e.g. brake-by-wire, video streaming, drones, etc. From a safety point of view, those applications can be divided into two categories: safety critical or non-safety critical applications. Safety critical systems are systems whose actions can decide over life, injury or even death. Non-safety critical systems are often infotainment systems.

A safety critical system, as any system, will fail once in a while. Such a failure can have many causes: a software bug, resistor short-circuit, rising temperature, etc. In this paper, the soft error is presented as a cause of failure. A soft error is a disturbance of hardware caused by external factors which affect the software in execution. These external factors can be radiation, fast temperature increase, etc. If the software is not aware of the occurrence of a soft error, the program memory can be corrupted leading to unstable behavior and unpredictable states. When using critical hardware components this can possibly lead to very dangerous situations.

The next two sections will discuss the two main domains of the research, respectively functional safety and soft errors. Then we will present some links between the two domains and use those links to join the domain of soft errors with that of functional safety. To conclude, future work is presented and conclusions are drawn.

II. FUNCTIONAL SAFETY

Embedded systems used as safety critical systems must comply with a functional safety standard, whose goal is to reduce the risk of physical injury or damage to health of people either directly or indirectly. This section will discuss the generic functional standard IEC 61508 and some differences made by more domain specific functional safety standards.

J. Vankeirsbilck, H. Hallez and J. Boydens are with:
1) "ReMI", Faculty of Engineering Technology, KU Leuven Zeedijk 101, B-8400 Oostende, Belgium;
2) "DISTRINET", Dept. of Computer Science, KU Leuven Celestijnenlaan 200A, B-3001 Leuven, Belgium, e-mail: {Jens.Vankeirsbilck, Hans.Hallez,Jeroen.Boydens}@kuleuven.be

2.1 Demystifying IEC 61508

The most generic functional safety standard is IEC 61508. As described by [1] and [2], it is an "umbrella" document covering multiple industries and applications. Its basic purpose is to create requirements intended to achieve reliable systems that either work properly or fail in a predictable manner. It is based on two concepts on which we will elaborate in the following sections:

- The Safety Life-Cycle, a detailed engineering design process which reduces failures due to systematic errors.
- The Safety Integrity Levels, quantified levels of failure rates.

2.1.1 Safety Life-Cycle

The Safety Life-Cycle is a model to eliminate systematic errors. IEC 61508 defines systematic errors as errors that lead to failures that are deterministically related to a certain cause. They are typically design mistakes. Software errors are considered systematic errors.

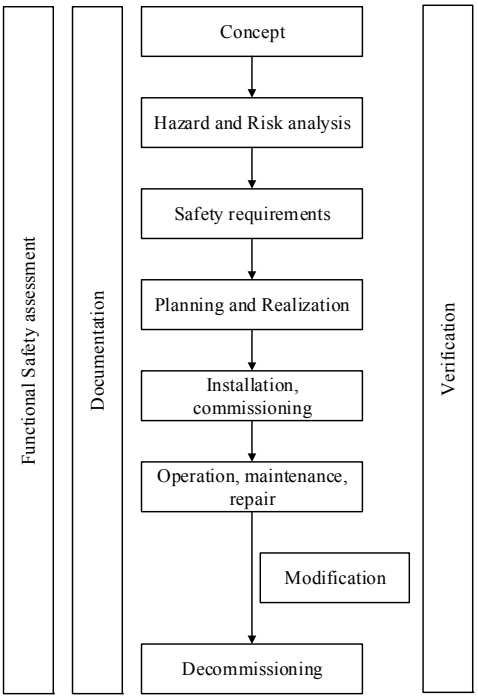

Fig. 1. Safety Life-Cycle

The Safety Life-Cycle addresses these errors by providing a classical, step-sequential waterfall engineering process. As Fig. 1 shows, it starts with the concept of the product and ends at its decommissioning. Each phase has to be documented and verified before moving to the next one.

2.1.2 Safety Integrity Level

Next to systematic failures, IEC 61508 also addresses random hardware failures. These are failures attributable to specific components and can be quantified. These quantifications result in a failure rate. The failure rates of components can be found in external or in-house failure rate databases or can be estimated using mathematical models. Once known, they help to predict the performance of the system.

IEC 61508 uses the concept of Safety Integrity Levels (SIL) to express excellence of the performance of components and the systems. The idea is to divide the "spectrum" of integrity into four discrete levels and then lay down requirements for each level. The higher the SIL, the more stringent the requirements. For IEC 61508 (and many other standards) the four levels are given in Table 1.

TABLE 1 SAFETY INTEGRITY LEVELS

SIL	High Demand (dangerous failures/hr)	Low Demand (PFD)
4	$\geq 10^{-9}$ to $< 10^{-8}$	$\geq 10^{-5}$ to $< 10^{-4}$
3	$\geq 10^{-8}$ to $< 10^{-7}$	$\geq 10^{-4}$ to $< 10^{-3}$
2	$\geq 10^{-7}$ to $< 10^{-6}$	$\geq 10^{-3}$ to $< 10^{-2}$
1	$\geq 10^{-6}$ to $< 10^{-5}$	$\geq 10^{-2}$ to $< 10^{-1}$

A distinction between high demand and low demand systems must be made. For high demand systems, e.g. car brakes, it is necessary to know the failure rate per hour since there is a high probability of suffering the hazard immediately each failure occurs. High demand systems are defined as systems that are used more than once per year. Low demand systems are used less than once per year. For low demand systems the failure rate alone is of little use since the hazard is not incurred immediately each failure occurs. The demand of the system is infrequent, so failures may well be dormant. Therefore low demand systems are characterized using the probability of failure on demand (PFD), it is necessary to know the chance of failure when you actually need the system.

2.2 Specific Industries

To compensate the generic character of IEC 61508, many industry domains have created their own functional safety standard, starting from IEC 61508.

- Medical: the medical domain has a general functional safety standard: IEC 60601 [3], which describes all requirements to build a complete medical system. Recently, IEC 62304 has emerged as a standard to describe the software development processes for medical systems. A difference between IEC 61508 and IEC 60601 is that IEC 60601 does not use SIL to categorize different medical systems.
- Railway: the railway domain has three standards, EN 50126 which describes Reliability, Availability, Maintainability and Safety; EN 50128 [4] describes the software development processes; and EN 50129 describes System Safety. A difference between the EN 5012X and IEC 61508 is the introduction of Software Safety Integrity Levels (SSIL) by the EN 5012X series.

III. SOFT ERRORS

This section discusses the second domain of our research: the soft error. First the main causes of the soft error are presented and secondly the possible effects of a soft error on the embedded system are listed.

3.1 Cause

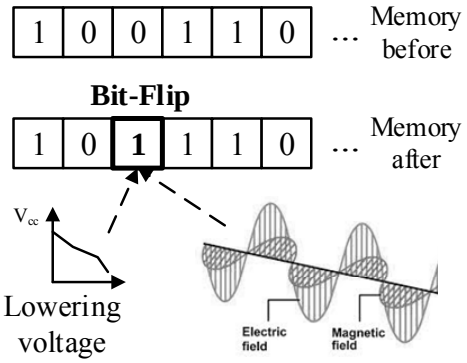

Fig. 2. Cause and Effect of a Soft Error

Fig. 2 depicts the two main causes of soft errors: the manufacturers, symbolized by the graph 'lowering the supply voltage', and the work environment, symbolized by the electromagnetic wave.

One cause of a soft error is the current trend followed by manufacturers. To satisfy their customers, manufacturers of embedded systems or components used by embedded systems have decreased the supply voltage, decreased the size… of those systems. While this has many advantages, e.g. increased battery lifetime, increased transistor density so the same die area can house more computing power… it also has one big disadvantage: the devices are now much more susceptible to soft errors.

Another cause of soft errors is the work environment of the embedded system, which is nowadays filled with radiation emitted from other electrical appliances, WiFi hotspots, natural background radiation, contaminated packaging, etc.

The radiation leads to highly energized particles that can pass through the embedded system, affecting its hardware. As described by Baumann [5], the passing particle can create a short pulse which can flip a transistor from open to close or vice versa. Due to the decreasing scale and supply voltage of the embedded system, the energy needed by the particle to flip the transistor has decreased, leading to a higher occurrence of soft errors.

3.2 Effects

Fig. 2 shows the primary result of a soft error: a bit of a memory location is flipped. That faulty bit does not necessarily have an effect on the system.

Fig. 3 is a simplified adaptation of the flowchart provided by Mukherjee [6] and shows how certain conditions have to hold before the bit-flip affects the system or the software in execution.

The first condition is that the faulty bit has to be read. Since soft errors are transient errors, overwriting the bit will discard the previous value, masking the bit-flip.

Secondly the bit has to be an Architectural Correct Execution or ACE bit. An ACE bit is a bit necessary for correct execution, e.g. a bit from the program counter. If a bit is un-ACE, thus unnecessary for correct execution, it cannot affect the system or software. An example of an un-ACE bit is a bit of the branch predictor. A branch predictor is a structure that tries to predict which branch has to be taken based on previous decisions. If a bit is flipped in this structure, or any other prediction structure, the performance of the system may be affected, but it will not affect correct execution.

If both conditions hold, the faulty bit is read and it is an ACE bit, the bit-flip will affect the software in execution. The effect on the software largely depends on the location of the bit-flip: Main Memory, General Purpose Register, Program Counter or Stack Pointer. Generally speaking, the effects can be divided into two categories:

- Data flow corruption, leading to wrong intermediary and output values.
- Control flow corruption, affecting the execution order of the instructions. A control flow error leads to a jump to unintended instruction.

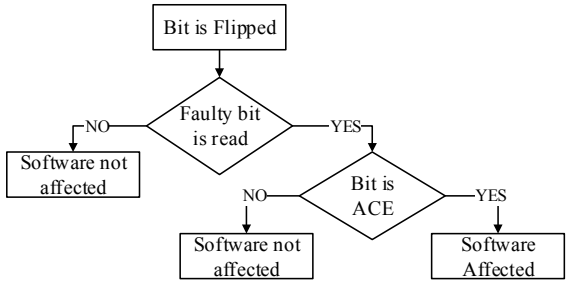

Fig. 3. Simplified flowchart to determine if the Bit-flip will affect the software.

Embedded systems are often used to control parts of its environment, an actuator, a servomotor... through I/O. This I/O is often memory mapped in the Main Memory of the embedded system. A bit-flip affecting a memory mapped I/O register may not affect the software in execution, but can affect the environment, e.g. controlling an actuator when it is actually prohibited. This can create very dangerous situations!

IV. INTEGRATION

To be able to join soft errors and functional safety, soft errors must be classified as systematic or random hardware failure. The failure to take into account is not the failure of the software, but the bit-flip since the software only fails due to the occurrence of a bit-flip.

To be able to classify the bit-flip, we took a look at the definitions for the failures given by IEC 61508:

- A systematic failure is a failure related in a deterministic way to a certain cause, which can only be eliminated by a modification of the design or of the manufacturing process, operational procedures or other relevant factors.
- A random hardware failure is a failure occurring at a random time, which results from one or more degradation mechanisms.

By analyzing the definitions, we conclude that a soft error is a systematic error. The bit-flip can always be traced back to a certain cause, be it e.g. EMI or rising temperature, and it can only be eliminated by a design modification.

4.1 Reducing the failure rate

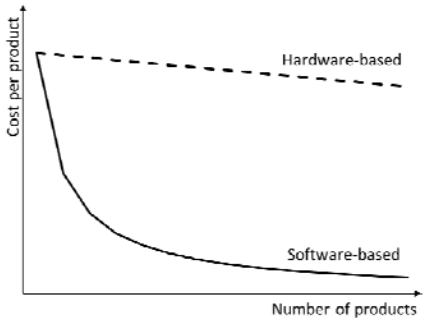

Fig. 5. Cost of soft error detection and recovery

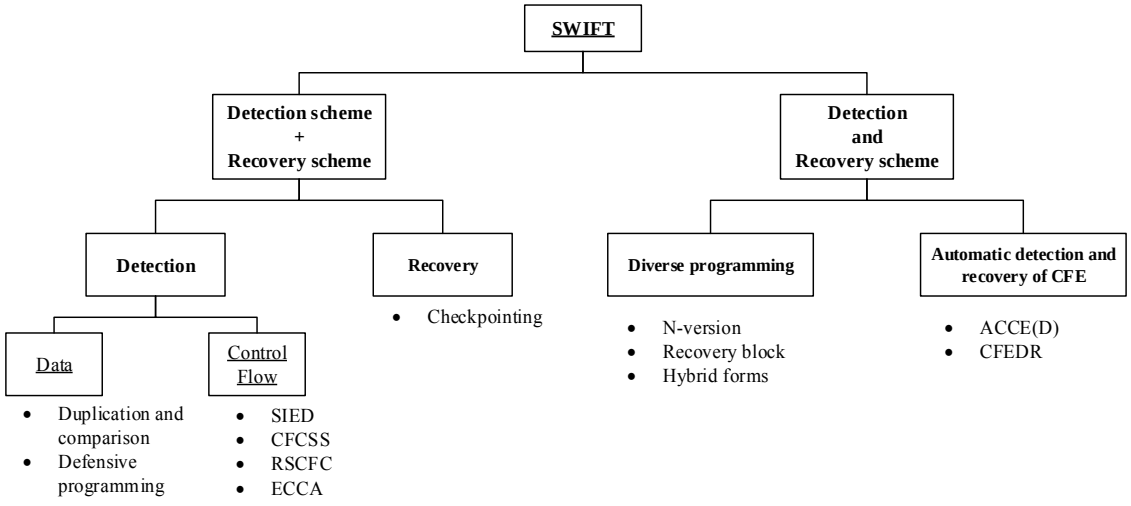

Fig. 4. Software implemented fault tolerance techniques

Functional safety is about reducing the risk of system failure. Soft Errors play a role in the failure rate of the device, so detecting and recovering from them will reduce the rate. Adding a detection and recovery technique reduces the number of soft errors that have an effect on the software, resulting in a reduced failure rate for the system.

Detecting and recovering from soft errors can be hardware-based or software-based. As Fig. 5 shows, detecting soft errors through software is more cost efficient. Hardware-based means adding components, e.g. a shield, to the PCB, which means this must be added for each product. This represents a nearly constant cost per product. Software-based detection techniques represent a high cost at development time, but once the software is completed it can be applied to each product, lowering the cost significantly for each product sold. Therefore future research will focus on software implemented detection and recovery techniques.

A second reason to focus on software implemented techniques is to ease compliance with functional safety standards. IEC 61508 states that: System software should include software for diagnosing faults in the system hardware, error detection for communication links and online detection of the application of software modules. The software in execution should be tested for control and data flow. Since data and control flow corruption are effects of a soft error, adding a software implemented detection and recovery technique ensures those corruptions are detected, and thus helps compliance with IEC 61508.

V. Future Work

Future research will focus on software implemented detection and recovery techniques or software implemented fault tolerance (SWIFT). Fig. 4 gives an overview of existing SWIFT techniques. The techniques have been grouped by their ability. The left group of techniques all have their specific function, e.g. SIED is meant to detect control flow errors and duplication is used to detect data flow errors. The right group of techniques detect and recover from the effects of the soft error, e.g. the diverse programming techniques use multiple versions of the program and a voter to decide on the correct result, masking the effect of the soft error.

First we will further complete the list by performing a literature study.

Once the list of existing SWIFT techniques is more or less complete, research will focus on implementing the techniques while complying with the rules of functional safety. Although soft errors and its detection techniques can theoretically be joined with functional safety, research will have to determine if this can be translated in practice. Software implemented techniques must comply with the software rules imposed by functional safety. Functional safety standards demand the use of a programming language subset, to force the programming language to be fully and unambiguously defined. Moreover, the standards demand the use of a coding standard and demand the limitation of certain programming constructs, e.g. pointers. These rules could make it impossible or very hard to implement existing soft error detection and recovery techniques.

VI. Conclusion

This paper discussed how to join the domain of functional safety and soft errors. This is done by proving that a soft error is a systematic error and can only be eliminated by a design modification.

The two domains are also joined since adding a soft error detection and recovery technique helps comply with IEC 61508, the most generic functional safety standard. IEC 61508 states that the system software should have diagnostics to be able to detect malfunction. Soft error detection techniques are in fact diagnostic techniques, so compliance with the standard is eased by implementing a detection technique.

References

[1] D. J. Smith and K. G. Simpson, SAFETY CRITICAL SYSTEMS HANDBOOK: a straightfoward guide to functional safety, IEC 61508 (2010 edition) and related standards, including process IEC 61511 and machinery IEC 62061 and ISO 13849, Third ed., Elsevier, 2010.

[2] M. Medoff and R. Faller, Functional Safety: An IEC 61508 SIL 3 Compliant Development Process, 2nd ed., exida, 2012.

[3] *IEC 60601-1: General Requirements for Basic Safety and Essential Performance,* 2005.

[4] *EN 50128: Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems,* 2011.

[5] R. C. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *Device and Materials Reliability, IEEE Transactions on,* vol. 5, no. 3, pp. 305-316, 2005.

[6] S. Mukherjee, Architecture design for soft errors, Morgan Kaufmann, 2011.

[7] J. Autran, D. Munteanu, P. Roche and G. Gasiot, "Real-time soft-error rate measurements: A review," *Microelectronics Reliability,* vol. 54, no. 8, pp. 1455-1476, 2014.