SRAM-based FPGA Systems for Safety-Critical Applications: A Survey on Design Standards and Proposed Methodologies **DRAFT**[*]

Cinzia Bernardeschi[1] Luca Cassano[2]
Andrea Domenici[1]

[1]Department of Information Engineering, University of Pisa, Italy

[2]Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy

Abstract

As the ASIC design cost becomes affordable only for very large scale productions, the FPGA technology is currently becoming the leading technology for those applications that require a small scale production. FPGAs can be considered as a technology crossing between hardware and software. Only a small number of standards for the design of safety-critical systems give guidelines and recommendations that take the peculiarities of the FPGA technology into consideration. The main contribution of this paper is an overview of the existing design standards that regulate the design and verification of FPGA-based systems in safety-critical application fields. Moreover, the paper proposes a survey of significant published research proposals and existing industrial guidelines about the topic, and collects and reports about some lessons learned from industrial and research projects involving the use of FPGA devices.

[*]Submitted to the *Journal of Computer Science and Technology.*

1 Introduction and Motivations

Since the first FPGA device was developed by Xilinx in 1984 with the XC2064 chip, the FPGA technology has enormously grown in terms of flexibility, reliability and computational power. Although it is still not comparable with ASIC technology either in terms of computational power or silicon area occupation, the FPGA technology has imposed itself in many application fields thanks to very good performance, low non recurrent design cost and very short time to market.

In particular, SRAM-based FPGA devices are employed in many application fields such as broadcast, wireless and wired communication systems [15], cryptography and network security [49], and consumer products, as well as in fields with stringent safety requirements, such as airborne [19], aerospace and defense [47], railways [21], and industrial and nuclear power plant control [55].

This interest is due to the capability of SRAM-based FPGAs of being dynamically and partially reconfigured at run-time, which makes this technology much more powerful and flexible than non dynamically reconfigurable technologies, such as flash- and antifuse-based FPGAs. Using SRAM-based FPGAs and exploiting dynamic partial reconfiguration, a designer can adapt the functionality implemented by the system to changing environment and operational requirements. For example, dynamic partial reconfiguration has been used in a platform for satellite payload processing [60]. A satellite payload may perform different tasks in the course of its mission, such as acquiring data, process it, and transmit it to ground. FPGA devices may be reconfigured for each task, thus improving resource utilization.

Nevertheless, SRAM-based FPGA devices are still seldom used in those parts of systems related with the safety of the system itself, due to the vulnerability to faults of the SRAM-based configuration memory [61]. On the other hand, in the last years a number of dedicated conferences and workshops, such as the NASA/ESA Conferences on Adaptive Hardware and Systems and the Military and Aerospace Programmable Logic Devices Workshops demonstrate great interest in employing SRAM-based FPGA devices in safety- and mission-critical applications. In [53], the maturity of reconfigurable FPGA technologies for safety-critical applications is discussed.

A safety-critical system is a system whose failure or malfunction may result in death or serious injury to people, loss or serious damage of equipment, or environmental harm. In the IEC 61508-2 functional safety standard [42], for safety related Electric/Electronic/Programmable Electronic systems (E/E/PE) operating in a low-demand mode of operation, the lower limit on the target failure measures is set at an average probability of 10^{-5} dangerous failures per hour of functioning. On the other hand, for E/E/PE safety-related systems operating in a high-demand continuous mode of operation, the lower limit is set at an average probability of 10^{-9}.

As discussed in [13], testing alone cannot guarantee such requirement; combining fault tolerant approaches, such as replication and diversity, together with testing and other techniques such as Failure Mode and Effects Analysis (FMEA) and reliability analysis methods, can improve the reliability of the system, but the result is still far from the 10^{-9} goal. It is then necessary to complement the *fault removal* (i.e., testing) and *fault tolerance* strategies with a *fault avoidance* strategy, with the goal of producing high-quality systems, as free as possible of systematic faults. This goal can be achieved with rigorous development processes carried out according to standards that explicitly take into account the requirements of safety-critical systems.

Many standards are available to developers of safety-critical systems, but most of them do not directly address the specific issues of the FPGA technology, or provide only limited guidance about them. Two are the main differences between the ASIC and the FPGA design from the system designer point of view. The first difference is that the FPGA design flow is much more automated than the ASIC one, and thus it leads designers to rely much more on the CAD tools provided by the FPGA vendor and to pay less attention to verifying the correctness of the intermediate products of the various design phases and to trust too much the CAD tools [22]. The second difference is that the final product of the FPGA design is a software, i.e., the bitstream. Because of this, FPGAs are often perceived by designers as easy to modify and correct late in the development process, thus FPGA based systems are often designed with development methods more similar to a *code and fix* approach than a true hardware design process, methods that would not be accepted for the design of more costly and less flexible technologies, such as ASICs or microprocessors [18].

In [18] Cercone *et al.* discuss how FPGA programming has not evolved much beyond the classical sequential development methodology of specifying requirements, creating the design, coding, simulating and testing. Often the documentation and testing of an FPGA project is left as an "end of project" task. The authors discuss how logic and functional testing are often completed only for known operational conditions, thus ensuring that the device does what it is supposed to do, but without ensuring that it does not perform unrequested functions. The paper strongly endorses the necessity of adapting verification and validation methodologies relying on modern design processes to the FPGA design, incorporating verification techniques as integral parts of the entire design process.

Habinc *et al.* [35], as well as Fernández-León in [22] and Gibbons and Ames in [29], discuss how many problems and failures in space applications involving FPGA devices are the result of applying inadequate development, verification and validation methodologies. The authors observe that since the FPGA technology became sufficiently mature, it is being employed more and more heavily in space applications, performing more and more complex and critical tasks.

Gibbons and Ames discuss the failure of the NASA Wide Field Infrared Explorer (WIRE) project, that was due to the indeterminate state of the output of a control FPGA device, during the power-up phase. The authors focus on that experience, arguing that a robust design process of an FPGA-based safety-critical system must rely on a great experience of designers in any aspect of the specific FPGA technology employed.

Finally, in [22], Fernández-León discusses the results of an audit of FPGA-based designs conducted by the European Space Agency, which revealed that the overall design methodology and quality control applied to these designs were often poorly defined and in some cases even risky or negligent.

Taking these issues into account, designers of FPGA-based systems often borrow standards and guidelines from more traditional technologies and adapt them to the needs of FPGA-based development. Moreover, because of the lack of specific regulations and standards, a number of guidelines, such as [1, 7, 56], and lessons learned from research and industrial projects, such as [18, 22, 29], have been published over the years.

Our work intends to present a brief overview of the existing standards for the use of FPGAs in safety-related systems, and, in general, hardware-based systems development, and to survey proposed techniques, guidelines and lessons learned about the design, verification and validation of FPGA-based safety-critical systems. This work is meant for both practitioners and researchers working in the field of design and verification of FPGA-based safety-critical systems. In particular, practitioners could exploit the present work to get a quick overview of the existing standards as well as to enrich their background through lessons learned from industrial and research projects involving the development of FPGA-based systems. On the other hand, researchers approaching the novel design and verification techniques could obtain from the present work a first picture of the existing trends.

Since the design of an FPGA-based system has many steps in common with the ASIC design flow and since there is not yet a comprehensive and specific standard for the development of FPGA-based systems, in the following sections we report general information, requirements and guidelines specific of ASIC designs but also applicable to FPGA designs, and, when available, activities and requirements specific of the FPGA design flow.

The remainder of this paper is organized as follows: in Section 2 we briefly describe the main features of the FPGA technology; in Section 3 we quickly review the standards in force for FPGAs in safety-critical application fields; in Section 4, we present a survey of the research proposals, guidelines and lessons learned for FPGA-based system design and verification in safety-critical application fields; Section 5 presents the main techniques for the analysis of the effects of radiation on SRAM-based FPGA systems; Section 6 reports on some published case studies; Section 7 discusses open issues; finally, Section 8 concludes the paper.

2 The FPGA Technology

An FPGA is a prefabricated array of programmable blocks, interconnected by a programmable routing architecture and surrounded by programmable input/output blocks. Figure 1 shows the basic architecture of an FPGA chip.

Programmable blocks may be simple combinatorial logic (*Soft Logic Blocks*) or memories, multiplexers, ALUs and other kinds of prefabricated circuitry (*Hard Logic Blocks*). Logic blocks may be programmed to implement a certain functionality, the routing architecture may be programmed to interconnect various blocks, and I/O pads may be programmed to ensure off-chip connections.

The purpose of the logic block is to provide the basic computational and storage element for the construction of the complete logic system. The programmable routing architecture, composed of wires and programmable switches, provides connections among logic blocks and I/O blocks to complete a user-designed circuit. Finally, the I/O architecture is composed of I/O pads disposed along the perimeter of the FPGA device, each one implementing one or more communication standards.

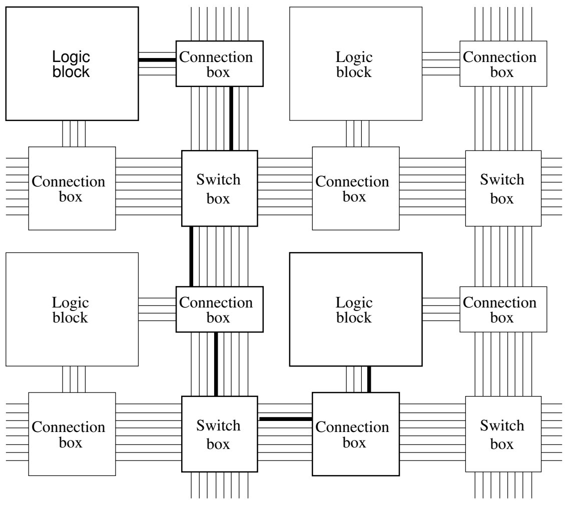

Figure 1: Basic FPGA structure.

FPGA programming consists in downloading to the device a programming code, called bitstream, that directly defines the hardware structure of the FPGA device by enabling or disabling gates in logic blocks to implement a certain function and in enabling or disabling connections between wires to connect or disconnect logic blocks, or to connect or disconnect logic blocks to/from I/O pads.

Three FPGA programming technologies exist: static memory (SRAM) based, non-volatile memory (flash and EEPROM) based and antifuse based [46].

In the static memory based programming technology, the bitstream is downloaded in the *configuration memory* of the device. This technology allows an indefinite number of device reprogrammings and has the best ratio of the device area for user resources to the area used by configuration memory, because static memory is realized with the same standard CMOS used for FPGA devices. On the other hand, SRAM-based FPGAs need a supporting non-volatile memory to store the configuration data while the device is not powered. Further, FPGA devices based on SRAM are the most susceptible to the adverse effects of radiations.

The flash/EEPROM programming technologies grant a limited number of reprogramming processes and have a worse real-estate utilization, because flash/EEPROM cells are fabricated with a non standard CMOS production process. On the other hand these technologies do not need any supporting non-volatile memory. Further, flash/EEPROM-based FPGAs are much less susceptible to the effects of radiations, and, in particular, almost immune to single event upsets in the configuration memory.

The antifuse programming technology does not allow any reprogramming of the device. Like flash/EEPROM-based FPGAs, also antifuse-based ones have bad real-estate utilization, but, again, they do not need any supporting non-volatile memory. Finally, antifuse-based FPGAs have no configuration memory, and are thus immune from long-term effects of SEUs.

The FPGA device performance both in terms of computational speed and silicon area occupation is proportional to the size and complexity of the basic logic block, but the simpler is the structure of the logic block itself, the higher is the degree of flexibility and programmability offered to the designer, so a trade-off between performance and flexibility must be found. Likewise, the higher is the number of switch boxes and long wires in the routing architecture and the number of communication standard implementable in a single I/O pad, the higher is the flexibility level, but at the same time, the worse are the computational speed and area occupation performance.

Modern FPGAs are extremely complex and powerful devices. They can be configured to host a complete microprocessor, or even a *System-on-Chip*, i.e., a complete system, composed of processor, memory and peripherals, all placed on the same chip. Many embedded processors that can be placed on FPGA devices exist, among which we can mention the Xilinx MicroBlaze and PicoBlaze, and the Altera Nios and Nios II, provided by the FPGA vendors themselves. Further, apart from the previously mentioned soft-CPUs, more complex and powerful cores, such as the ARM Cortex-M1 core or the Gaisler LEON4 CPU, can be placed on modern, high-performance, FPGA devices.

More detailed discussions about FPGA architectures can be found in [46].

2.1 Advantages and issues related to FPGA-based designs

As has been analyzed in depth by Kuon and Rose [45], FPGA-based designs are usually larger, slower and much more energy-consuming than full-custom designs. Nevertheless, they are more and more widely employed in all application fields and the interest in using FPGA devices in safety-related applications, such as space missions or railways systems, is growing. This is basically due to the two main factors of low costs and short time-to-market [46].

A full-custom design needs very expensive CAD tools for simulation, verification, synthesis, and floor-planning. Further, a full-custom design needs a large number of engineers working for many months. Finally, full-custom designs require the use of masks, which may cost several millions dollars, to drive the lithographic process. On the other hand, the cost of an FPGA device ranges between some tens and one thousand dollars. Further, licenses of CAD tools for FPGA-based designs are much cheaper than those of tools for full custom designs. The only large cost related to FPGA-based designs is the cost of the development team. Thus, especially for low-scale, but also for medium-scale productions, FPGAs are often the best technological choice.

From the time-to-market point of view, after the completion of a full custom design process, the designing company must send the obtained masks to a silicon foundry (often located in a different country or even continent) to physically produce the chips, which may be sent back to the customer up to three months later. Then, the chips have to be tested and, if modifications to the design are needed, the design process has to iterate one of the previously performed design activities, after which a new fabrication process will be performed. A full custom design may need up to three fabrication iterations and thus up to twelve or even eighteen months between the product conception and its availability to customers. FPGA-based designs, instead, do not require fabrication delays. Design errors can be identified much more easily during the prototyping phase and thus the time-to-market of an FPGA-based design generally ranges between three and six months.

The main issues related to the design of FPGA-based systems and to their adoption in safety-critical application fields are the lack of standards specifically addressing the FPGA technology and the severe susceptibility of FPGA devices to the effects of radiations.

As we have already discussed in the introduction of this paper, given the relative youth of the FPGA technology, and its not yet wide acceptance in safety-related application fields, very few design standards specifically addressing the FPGA technology exist. This, together with the ease and the low cost of prototyping and fixing defects, often led designers to underestimate design issues, thus causing projects failures. In order to overcome the lack of standards and to guide designers, several industrial guidelines and lessons learned from projects have been published.

Radiations, both in space and at ground level, may cause a system to fail. In particular, radiations hitting the silicon surface of digital circuits may alter the content of memory elements. Radiation hitting SRAM-based FPGA devices may have even worse effects, since they could permanently corrupt the contents of the configuration memory (until a device reconfiguration), thus changing the functionality implemented in the device.

3 Standards regulating the design of Hardware Systems in Safety-Critical Systems

A general framework for the design and development of hardware and software safety-critical systems is the *IEC 61508* standard, and in particular the *IEC 61508-2* [42] and the *IEC 61508-3* [43] for hardware and software systems respectively. Currently a specific regulation on FPGA design in safety-critical systems exists only in the aerospace application field: the *ECSS-Q-ST-60-02C* [26]. Moreover, in the airborne application field, the *RTCA/DO-254* standard [23] can be applied to the design and development of electronic systems and thus also to FPGA-based systems. In all the other application fields, the in force regulations require adopting the standard for the design and development of both software and hardware systems. In particular, in the automotive application field the reference standards are the *ISO/DIS 26262-5* [27] and the *ISO/DIS 26262-6* [28]; in the railways application field the *CENELEC EN 50128* [24] and the *CENELEC EN 50129* [25].

Since the design of modern VLSI systems always involves the use of high level hardware description languages, all the standards agree on the application of a V-shaped design flow, which is inspired by the classical software design flow. Figure 2 shows the V-shaped design flow taken from the *IEC 61508-2* [42] standard for the ASIC development lifecycle. Note that, although the figure representing the V-shaped lifecycle seems to be specifically related to ASIC designs, the same lifecycle has to be taken into account also when the design of an FPGA-based system is addressed [42].

In the V-shaped lifecycle, in each phase of the development (the left-hand branch of the V) requirements are defined and sub-products are produced to fulfill the requirements. Moreover, after each phase the intermediate products of the phase are verified against the requirements specified in the previous phase: for example, the adequacy of the hardware architecture in fulfilling the requirements specification must be verified, the adequacy of the designed modules and their integration in fulfilling the architecture must be verified and so on.

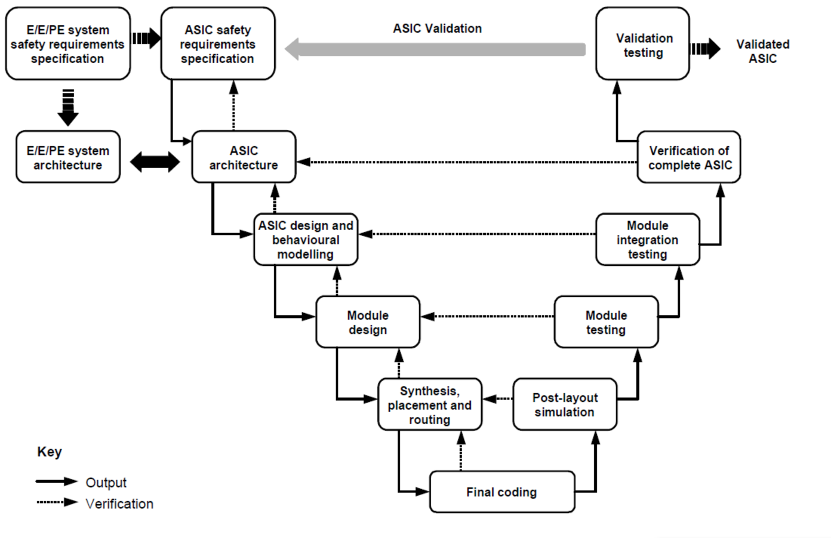
E/E/PE system safety requirements specification
ASIC safety requirements specification
ASIC Validation
Validation testing
Validated ASIC
E/E/PE system architecture
ASIC architecture
Verification of complete ASIC
ASIC design and behavioural modelling
Module integration testing
Module design
Module testing
Synthesis, placement and routing
Post-layout simulation
Final coding
Key
Output
Verification

Figure 2: ASIC development lifecycle (the V-shaped model) [42].

In the validation phase of the lifecycle (the right-hand branch of the V) all the products of the development phase are evaluated in order to ensure correctness and consistency with respect of the global requirements. In other words, it must be verified that the obtained FPGA-based system fulfills the functional and safety requirements specified in the requirement specification phase and does not contain undesired functionalities. In some cases, this final validation shall be carried out by an independent party.

The ECSS-Q-ST-60-02C standard requires that the system verification is performed in a realistic application environment. Thus, a system breadboard shall be designed and used covering all the operating modes and conditions of the device. Also radiation testing shall be performed on the prototype if the required radiation-hardening level is not yet granted by the used technology.

In the following of this section we present a brief summary of the requirements imposed by the previously mentioned safety standards, for each phase of the V-shaped design lifecycle, placing particular emphasis on the requirements imposed by the ECSS-Q-ST-60-02C standard for FPGAs.

3.1 Development Process

3.1.1 System safety requirements specification

In this phase, starting from the requirements specification document of the whole system (named "E/E/EP system safety requirement specification" in the figure), requirements for the FPGA-based system are extracted and analyzed. In particular it is recommended to identify those requirements that involve functionalities that allow the system to reach and maintain a given safety level, those functions that allow the system to detect, identify and handle faults and those functions related to performance- and time-critical operations.

The specification of the system requirements shall contain details relevant to the design, to achieve the safety integrity level and the required target failure measure for the safety function, as specified by the E/E/PE system safety integrity requirements specification.

In particular, the ECSS-Q-ST-60-02C standard imposes the following additional requirements related to the occurrence of faults due to radiation:

- error handling
- test device on ground and flight
- proof of required fault coverage during tests

Moreover, the standard imposes the production of a feasibility study in order to estimate the requested power consumption, speed and radiation tolerance. At the end of this phase a document that completely collects and defines the requirements for the FPGA-based system is produced. This document is required to be complete, unequivocal, clear and precise, verifiable and testable.

An interesting point is that all the standards highly recommend the use of semi-formal methods, such as logic/function block diagrams, sequence diagrams, data flow diagrams, and of formal methods, such as finite state machines, timed Petri nets, LOTOS, OBJ and Z, for the specification, analysis and verification of high-level system requirements.

3.1.2 System Architecture

In this phase the overall architecture of the system is defined. In particular, the high-level components that will compose the system are identified, the interfaces among them are specified and the input and output of the system are defined. Moreover, the decision on how to partition the system into its hardware and software components shall be taken during this design phase. A significant effort shall be paid in identifying a hardware architecture able to fulfill the previously defined safety requirements: for example, architectural-level fault-tolerance schemes are selected in this phase. Moreover, in this phase, it must be defined which components of the architecture will be developed from scratch and which ones will be purchased as third party intellectual properties.

Desired qualities of the produced architecture are modularity, testability, maintainability and low complexity. The produced architecture shall be verified according to the previously defined requirements. To ensure that the architectural design captures the information necessary to allow the subsequent development activities to be performed correctly and effectively, the architectural design shall be described with appropriate levels of abstraction by using semi-formal and formal notations.

3.1.3 System design and behavioral modeling

In this phase the previously defined architecture is refined into a number of sub-components. The high-level behavioral specification of these components is defined in this phase. All the standards agree in requiring the use of hardware description languages (behavioral VHDL/Verilog) to describe the behavior of the components and about the observance of coding guidelines. Furthermore, proven-in-use design environments and simulators shall be used.

3.1.4 Module design

During the module design the high level behavioral model of the design is translated into a structural description composed of the hardware modules in accordance with the architectural design. In this phase the use of behavioral/structural hardware description languages is highly recommended. Moreover, in this phase area, power consumption and timing constraints of the defined modules are specified. The ECSS-Q-ST-60-02C standard places particular emphasis on the definition of time constraints and of a detailed pin plan for FPGA designs.

After each module has been designed, it must be tested in order to determine if it is fit for use. The purpose is to verify the implementation by testing every possible operational mode of the module. Static analysis tools are used to facilitate this process.

After all modules have been designed and integrated in the complete system, integration testing shall be performed. In integration testing the separate modules will be tested together to expose faults in the interfaces and in the interaction between integrated components. Testing is usually black-box as the code is not directly checked for errors.

3.1.5 Synthesis, placement and routing

After the detailed design has been completed it must be synthesized so to generate the gate-level netlist implementing the system. During this phase, proven-in-use simulation, synthesis tools and technological libraries must be used.

In the placement and routing phase the synthesized netlist is placed on the chip and routing information is defined in order to meet the timing constraints. Moreover, the power and clock distribution is performed.

3.1.6 Final coding

In the FPGA programming phase the placed and routed design is translated into the programming bitstream, the FPGA device is programmed and the resulting prototype is tested. The design validation will be performed on the produced prototypes of the system.

3.2 Validation Process

After the development phase, the implemented design must be validated. The first validation step is the post-layout simulation. At this stage (i) estimated delays shall be verified; (ii) gate-level simulations, formal verification and static timing analysis shall be performed; (iii) key parameters such as voltages, noise, frequencies, bandwidth, power consumption, shall be verified; (iv) functional verification shall be performed.

Moreover, the ECSS-Q-ST-60-02C standard asks to verify the effectiveness of the implemented radiation-hardening mechanism. Finally, the standard places particular emphasis on the use of IP-cores: when such modules are purchased and used, great attention must be paid in the verification of the IP-core itself and in the verification of the correct integration of the IP-core into the architecture under design. In particular it must be verified that the third-party IP-core exactly performs the functionalities declared by the vendor and does not implement hidden unwanted functionalities.

Post placement and routing verification shall be performed: electrical properties and cross-talk sensitivity shall be evaluated; I/O timing and power distribution shall be checked; it must be verified whether the obtained post place-and-route netlist is functionally consistent with the gate-level netlist; timing performance shall be evaluated; clock skew and clock latency shall be estimated.

Complete system testing will compare the system specifications against the actual system implementation. After the integration test is completed, the next test level is the system test. In the lower-level tests, testing is done against technical specifications, while system tests look at the system from the perspective of the customer and the future user. The testers validate whether the requirements are completely and appropriately met. Many functions and system characteristics result from the interaction of all system components, consequently, they are only visible on the level of the entire system and can only be observed and tested there.

4 FPGA research proposals, guidelines and lessons learned

In this section we propose a survey of research proposals, industrial and academic guidelines and lessons learned from real-world projects regarding the design and verification of FPGA-based systems in safety-critical application fields. We have organized the survey following the structure of the V-shaped lifecycle previously presented. For each phase of the design process we report activities that should be carried out as well as proposed techniques and tools.

4.1 Safety Requirements Specifications

In accordance to the standards, in [35] and [56] it is confirmed that the risk analysis should be carried out during the concept and requirements definition phase, along with the feasibility study and the requirements specification.

According to these two works, the feasibility study should: (i) Analyze the availability and quality levels of the existing FPGA technologies and identify a set of candidate FPGA target devices taking into account the amount of available logic blocks and I/O pins, the maximum reachable operating frequency and the power consumption level; (ii) conduct a preliminary worst case timing analysis of external interfaces and clock domains; (iii) check the availability and reliability of all required design tools and libraries.

The requirements specification document should discuss the following aspects of the design: (i) Identify details of the operating environment, such as temperature, humidity and dust; (ii) discuss safety-related requirements, such as self-test and auto-diagnostic capabilities, reliability, testability, maintainability and observability requirements and radiation-hardening level; (iii) identify requirements on the interfaces with external devices and the protocols that have to be followed between the device under design and any external devices; (iv) determine the power budget and the operating frequency range of the system; (v) identify the timing, size and electrical constraints of the system.

The risk analysis should identify the critical issues of the design and identify the possible backup solutions, including but not limited to: (i) Maturity of the foreseen FPGA device family, including CAD tools, libraries and vendor support; (ii) suitability of the chosen technology for the intended mission; (iii) undetermined I/O behavior and internal initial state during power-up.

In [22], Fernández-León stresses that a designer should implement a reliable development methodology for the definition, design, verification, physical implementation and validation phases. Moreover, the author points out that designers should assess and document the radiation threats to the circuit. The effects of radiation on the circuit shall be identified and countermeasures shall be properly designed, implemented and verified.

In [39], an innovative FPGA requirements specification process for the design of safety-critical systems based on programmable devices is presented. It is suggested that the design process should start with a requirements specification redacted using the formal specification language Z. Then, using the INFORMED design method, the boundary between hardware and software components of the system is identified. Any basic computation is then partitioned, by deciding what shall be implemented in hardware and what shall be implemented in software. For those functionalities that shall be implemented in hardware a manual refinement using *Synchronous Receptive Process Theory (SRPT)* is performed, and then each hardware module is compiled into

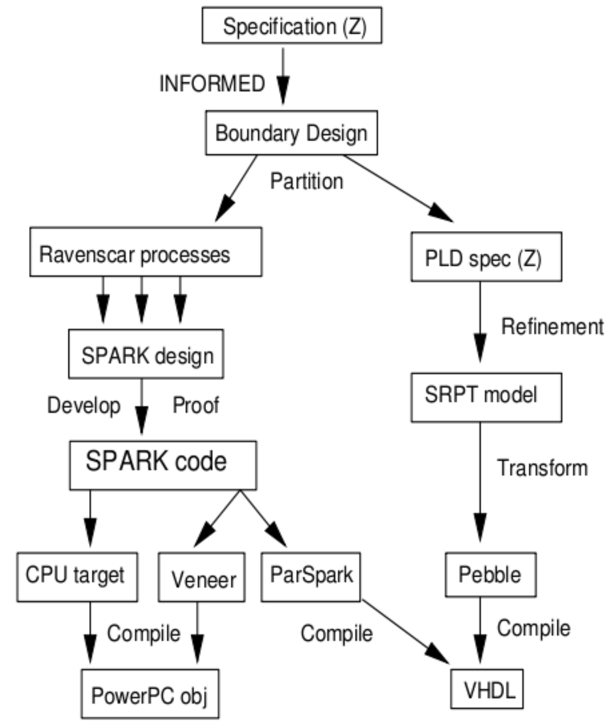

Figure 3: Development Process Proposed by Hilton *et al.* [39].

Pebble, and subsequently in VHDL. Software components shall be implemented in SPARK Ada and they will interact using the Ravenscar tasking subset. The main Ada program, and the FPGA device will then communicate through a veneer component, sending data to and from the FPGA via a method appropriate to the particular system, such as shared memory or dedicated bus. Figure 3 shows the overall development process proposed in [39].

In [61], Sutton underlines the need of machine-readable formalisms for requirements specification in order to guarantee that all the requirements have been addressed during the design process.

4.2 System Architecture

At this stage of the design life-cycle the target device shall be chosen and consequently the vendor's CAD tool shall be chosen and purchased. The choice of older and more stable and reliable CAD tool instead of newer ones, with better performance but lower reliability, is recommended. The use of hardware description languages, such as Verilog or VHDL, and of CAD tools to produce the architectural design is highly recommended [1].

The guidelines proposed in [35] and [56] give also suggestions about the architectural design phase.

In [35], Habinc gives recommendations about the architectural design phase of an FPGA employed in a safety-critical system, that were learned from the analysis of a great number of FPGA designs for space projects, many of which showed problems related with the FPGA device itself. In particular, Habinc focuses on four main issues: reset, clocks, power and interfaces.

While asynchronous reset allows an immediate reset of the flip-flop, it most often poses tight timing requirements on the routing of the reset signal. Because of this, the solution proposed in [35] is to assert the internal reset signal asynchronously and to de-assert it synchronously. For outputs that are critical for the system operation, it is recommended that the corresponding flip-flops are reset asynchronously. Finally, the state during and just after a reset should be documented in detail.

Clocks should control all storage elements, i.e, the design should be fully synchronous. The number of clock regions should be minimized since FPGA devices normally have only a few dedicated clock buffers. Since the choice between synchronous or asynchronous design is made at the HDL description phase, the use of simple HDL source code templates that are available from the FPGA vendors is highly recommended in order to avoid coding errors that would lead to an asynchronous architecture, while the desired architecture was synchronous, or viceversa. Clock gating techniques should be avoided.

Concerning power consumption, Habinc suggests avoiding clock signal manipulations that are in conflict with synchronous design methods. Careful attention to the power up and down sequences of FPGA devices should always be paid, since some technologies exhibit an uncontrollable behavior on their input and output pins during these phases. The power-up/power-on-reset sequence should be carefully defined and documented, following the vendor guidelines.

About I/O interfaces, Habinc suggests placing a Schmitt trigger inverter between the analog input sources and the FPGA inputs pins to reduce the risk of violating the rise and fall times. Moreover, Habinc recommends much attention in ensuring that bus contention cannot occur, internally as well as externally to the FPGA. FPGA devices have several special pins that are often not used by the application. Nevertheless, these pins need careful consideration during the design of the board on which the device will work. In general, it should be ensured that all special pins, test pins and unused pins are properly terminated, strictly following the FPGA vendor guidelines. Unused pins should normally be left unconnected. It is not recommended to connect unused pins directly to power or ground. Finally, the state of the unused pins shall be properly documented.

In [48], a large number of formalisms for high-level architectural system modeling are presented, such as Finite State Machines, Petri Nets and all their extensions, Statecharts and UML. Similarly, many languages and techniques for the verification of designed modules are discussed in [2]. The main approaches are: (i) The *e* language, an object-oriented language for testbench design, giving designers the chance to easily generate sets of input stimuli, specify constraints and properties and assess the simulation coverage; (ii) *OpenVera*, a testbench language similar to *e*, with a C-like syntax; (iii) *ForSpec*, a temporal logic based specification and modeling language developed at Intel; and (iv) *Property Specification Language (PSL)*, originally developed in 2004 by Accellera, and then standardized in 2005 by IEEE, which allows the designer to specify complex properties, combine them and then verify the final properties.

4.3 Behavioral Modeling and Module Design

In this phase a detailed description of the high level functional blocks defined in the previous phase shall be produced, implementing the defined functionalities, interfaces, interconnections and interactions [35].

In this phase the use of a hardware description language, such as Verilog or VHDL, and of CAD tools is also highly recommended [1]. A strict coding standard should be used to avoid systematic faults due to coding errors: it is suggested to avoid non-synthesizable code and coding instructions that would lead to the insertion of latches. The use of constants and parameters is highly recommended. Naming, indenting, spacing and commenting standards are useful to easily detect coding errors and to improve code understandability [56].

In [20], a VHDL guidance for safe and certifiable FPGA design is reported. In that paper Conmy *at al.* recommend avoiding states with encodings that differ by just one bit when designing finite state machines: in this way, a single event upset could not cause the machine to jump into an unwanted state. Moreover, they strongly suggest limiting the size of each VHDL module in order to improve module testability and maintainability. Finally, they suggest careful attention to the development of operations involving floating point numbers since they are particularly difficult to manipulate on an FPGA.

A very large number of alternative high-level hardware programming languages has been proposed as intermediate languages between the architectural design and the description of the device structure in a hardware description language. Most of these languages are derived from C, such as *Handel-C* and SPARK, from C++, such as Streams-C and ASC, or from Java, such as Sea Cucumber. A particular mention should be paid to SystemC, which is widely employed as a high-level design language for electronic systems.

Alternative approaches are: ELLA, which allows abstraction and formal reasoning about the design; Esterel, a synchronous language used for programming reactive systems, that can be automatically compiled in VHDL or Verilog; LAVA, a relational language designed to express circuit designs by describing the relative placement of building blocks of the circuit itself. Finally the MATCH compiler and the AccelFPGA compiler that allow translating MATLAB programs in VHDL and Verilog code for FPGAs.

After the implementation of all HDL modules the module integration phase shall be performed, gradually integrating the HDL modules in order to compose the whole system. Compliance to the required coding standards and guidelines must be verified [56]. Test benches shall be designed in order to perform behavioral simulations of each module, to check both external interactions and interfaces and internal data flows. For complex designs it is important to use self-checking test benches, that can perform the test activity, automatically check the results and produce a test report, without requiring a visual inspection of the waveforms. It is important at this phase to have automated test vector generators, in order to generate input sequences that can stimulate each part of the component at least once and as randomly as possible [56]. Also boundary value tests shall be performed in order to evaluate the robustness of the design.

In [22], the necessity of inspecting and simulating the synthesized netlist in order to verify its correctness is emphasized. Moreover, SEU simulation and emulation are recommended. Finally, it is required to check whether the designed fault tolerance techniques have been properly implemented and synthesized, without any unwanted extra logic insertion or redundant logic resources removal by CAD tools.

4.4 Synthesis, placement and routing, and final coding

A number of works presenting alternative place-and-route algorithms able to increase the robustness of a given design against faults have been published in the last years. In [58], the *Reliability-oriented Routing Algorithm (RoRA)* for TMR-based designs is presented. The work starts from the consideration that the XTMR tool from Xilinx fails in some cases to protect the design from single event upsets due to the presence of common causes of failure in the routing of the design. RoRA heuristically places and routes the three replicas of the design and the voting circuit in such a way that the four components of the design do not share any routing resource. The algorithm proposed in [64] tries to keep the length of wires as short as possible in order to reduce the likelihood of open/short faults and to reduce the common regions between two nets in order to reduce the likelihood of bridge faults. In [40], an alternative cost function for an existing place-and-route algorithm is presented. While legacy place-and-route algorithms try to optimize the timing or the area occupied by the design, the algorithm proposed in [40] tries to minimize the error propagation probability of the design. In [31], a place-and-route algorithm that aims at minimizing the number of configuration bits used by the routing resources of the design is presented. Finally, the problem of multiple cell upsets (MCUs) (bit flips of multiple configuration memory elements due to a single particle strike) in TMR-based designs is addressed in [57]. The paper presents PHAM, a placement algorithm that exploits the knowledge of the physical layout of the configuration memory of the device to maximize the distance between configuration memory cells belonging to different replicas of the design.

After the synthesis, place-and-route and bitstream generation, the correct functionality of the system coded in the bitstream shall be verified, as discussed in [22, 56]. Moreover, the correctness and trustworthiness of externally purchased IP-cores shall be assessed. Nevertheless, verifying the correctness of a system at the bitstream level is an extremely hard task. FPGA vendors do not provide any detail about the structure of the bitstream, and the problem of verifying third-party IP-cores is made harder by the fact that very often these cores are provided as obfuscated or encrypted netlists. Thus, designers generally perform testing activities on the programmed device, spending great effort in designing sufficiently effective test cases. Nevertheless, testing cannot be exhaustive for medium/large scale designs.

Recently, Luna Inc. developed a software platform called *Change Detection Platform* (CDP) [32]. This environment is able to reconstruct the logic and post-place-and-route netlists, as well as the behavioral description of the system, starting from the bitstream. In this way it is possible to verify that the translation tool provided by the FPGA vendor did not introduce bugs in the bitstream. Further, it is possible to verify whether the purchased netlist-level IP-cores do not contain defects, unwanted functionalities, or security flaws.

5 Radiation Effects Analysis and Mitigation

Radiations may produce system malfunctions [8]. In particular, radiations affecting digital circuits may cause changes in the contents of memory elements and in the value of signals. Radiations on SRAM-based FPGA devices have even worse effects, since, when affecting the configuration memory, they could permanently change the functionality implemented in the device (until reconfiguration) [33]. The above mentioned effect is known as Single Event Upset (SEU). Other effects of radiations on digital circuits are the *Total Ionizing Dose* (TID), i.e., accumulation of charge in the interface between the metal and oxide layers that cause an increase of power consumption and a decrease of circuit speed, and *Single Event Transients* (SETs), i.e., transient impulses on wires in the circuit. Neither TID or SETs have been widely studied in SRAM-based FPGAs since these devices are much more susceptible to SEUs, but they must be considered when other FPGA technologies are used [63, 52].

Although the effects of radiation are much more intense in space, it has been demonstrated that radiation may corrupt the behavior of digital circuits also at ground level [50]. The ECSS-Q-ST-60-02C requires that the radiation hardening techniques implemented in the design are assessed through radiation testing and the SEU sensitivity of the system is analyzed. Moreover techniques for mitigation of SEUs are either highly recommended or mandatory, depending on the safety level. A number of techniques have been proposed to this purposes. In the remainder of this section we will first present the main techniques for the analysis of the effects of SEUs on SRAM-based FPGA systems, and then some SEU mitigation techniques.

5.1 SEU Effects Analysis Techniques

The sensitivity to SEUs of SRAM-based FPGA systems can be analyzed according to four main approaches: *accelerated radiation ground testing*, *fault emulation boards*, *analytical computation*, and *fault simulation*.

Accelerated radiation ground testing [17] emulates the radioactive environment in which the system will work by exposing a prototype of the FPGA-based system to a flux of radiations. During the exposure to the radiation flux, the prototype is fed with a set of input patterns, and its behavior is monitored. The drawbacks of accelerated radiation testing are: (i) The impossibility of injecting SEUs only in the configuration memory of the FPGA, since the whole chip area will be irradiated (including user resources); (ii) a possibility that the device be permanently damaged after the experiment; and (iii) high cost.

A number of fault emulation boards has been developed to evaluate the effects of SEUs in the configuration memory of SRAM-based FPGAs systems [4, 3]. These boards emulate the occurrence of SEUs by modifying the bitstream of the target system whose behavior is then dynamically evaluated. Fault emulation can be performed either before downloading the bitstream on the device under test, or at run time exploiting partial dynamic reconfiguration. Unlike radiation testing experiments, fault emulation allows focusing specifically on SEUs in the configuration memory of the FPGA, leaving out any other resources. Moreover, fault emulation avoids the risk of damaging the device under analysis. The drawbacks of SEU *emulation* are: (i) high costs; (ii) complex usability; and (iii) strong chip and vendor dependence.

Analytical approaches, such as those presented in [59, 6, 38], have been developed to avoid the high cost of radiation testing and the long experimental time of fault emulation. In [59], a model based on the structure of the design implemented on the FPGA is built, and the topological modifications induced by SEUs in each configuration bit are deduced, thus discovering which SEUs affect the design. In [6], a model for the identification of sensitive paths to SEUs is presented. The model combines the error probability of all nodes of the circuit with the error propagation probability of each path of the circuit. Finally, in [38] a probabilistic model to estimate the reliability of SRAM-based FPGA system is presented. Given the probability of occurrence of a SEU, the model estimates the probability of having a system failure after a given amount of time. The drawback of these approaches is that, since the analysis is carried out without taking into account the input patterns fed into the system, they are able to provide a worst case analysis while they are not able to provide information about the behavior of the system in its normal operating conditions.

A large number of fault simulators for digital circuits can be found in the literature, but very few of them target the analysis of the effects of SEUs. Moreover, an even smaller number of simulators that specifically address the FPGA technology can be found. In [54, 14], two simulators of SEUs affecting digital circuits have been proposed. Both simulators work at the gate-level representation of the circuit, thus ensuring accurate results, but neither one takes into account any details specific of the FPGA technology. The only simulator targeting SEUs in FPGAs is *SST* [34] that works on the register transfer level representation of the system. Because of this, SST can only emulate the effects of SEUs in logic resources, e.g., flip-flops and memories, but it cannot reproduce the effects of SEUs in the configuration memory. Recently, a simulator of SEU effects in SRAM-based FPGAs, based on the stochastic activity networks formal specification language, has been proposed [11, 9].

5.2 SEU Mitigation and Correction Techniques

Many SEU mitigation techniques are discussed in the literature. In [44], SEU mitigation techniques are classified into two main families: *Fabrication process-based* and *Design-based*.

Fabrication process-based techniques aim at reducing the effects of radiation through the use of non standard CMOS logic gates, such as the *Silicon-on-insulator* (SOI) technology from IBM [41] and radiation-hardened memory cells [**?**]. The approach proposed by IBM in [41] relies on the placement of a thin layer of silicon on top of an insulator during the manufacturing process. All the transistors of the device are then built on top of this silicon layer, which is characterized by a reduced capacitance, and then by a reduced susceptibility to the effects of radiations. Radiation-hardened memory cells rely on providing standard memory cells with feedbacks devoted to restore the correct value when the content of the cell is corrupted. These techniques are able to alleviate the long-term effects of radiation exposure (i.e., TID), and to reduce, but not eliminate, SEUs. The main drawback of such solutions is the high cost due to the non standard CMOS fabrication process.

Design-based techniques rely on *hardware redundancy*, i.e., the use of extra components of the FPGA device (duplication or triplication) and voting systems to detect (when duplication is used) or correct (when triplication is used) the occurrence of SEUs [16]. A generalization of hardware redundancy is *device redundancy*, that is, using multiple independent FPGA devices performing the same functionality, whose output is then checked by a voting system. Design-based techniques are widely accepted because they are much cheaper than fabrication process-based techniques. Hardware redundancy exploits the spare components (when available) of the FPGA device, thus its cost (apart from the increased power consumption) is actually null. When device redundancy is used, the cost of multiple devices is always lower than the cost of non-standard CMOS devices. An additional advantage of design-based techniques is that they can be applied to different levels of design abstraction and can address different fault types.

Finally, SEUs may be corrected by exploiting the partial dynamic reconfiguration capabilities of modern FPGA devices through *readback and reconfiguration* [30] or *memory scrubbing* [37]. Both techniques are applied at run-time, during the system normal operation. Readback and reconfiguration consists in periodically reading, either partially or totally, the contents of the configuration memory, comparing it to a golden copy of the bitstream and reconfiguring the device in order to correct any detected fault. Memory scrubbing is similar to memory readback and reconfiguration, the main difference being that reconfiguration occurs at regular intervals. With *blind* scrubbing the whole bitstream is reloaded, irrespective of the occurrence of faults, whereas with *selective* scrubbing readback operations make it possible to identify faults and correct them with partial reconfigurations.

6 Case Studies

In this section we cite works that report on FPGA applications in different industrial fields and address development issues.

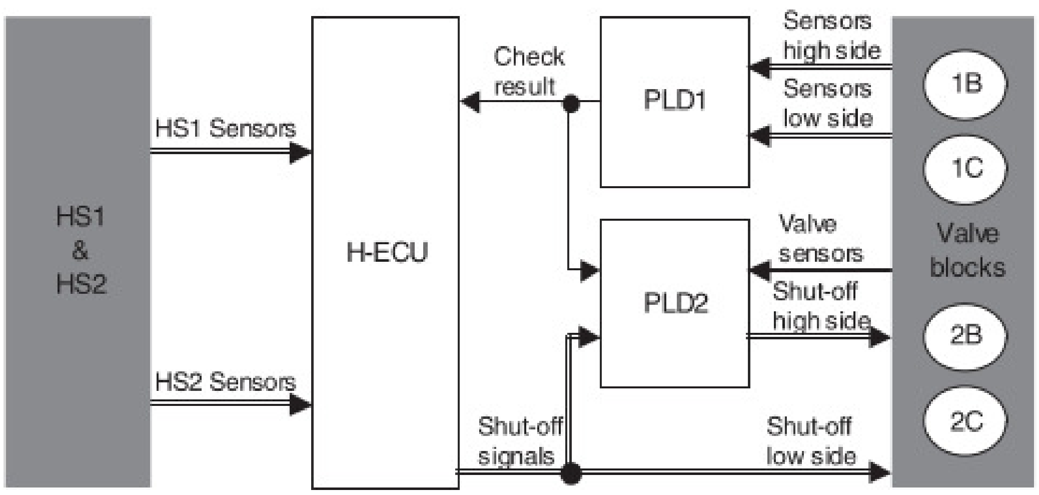

Figure 4: Hydraulic monitoring system (from [36]).

6.1 Hydraulic Leakage Monitoring

Hydraulic systems are used in aircraft to actuate highly critical components, such as control surfaces and landing gear. Leakages may cause pressure losses, which may lead to catastrophic failures, so a *Hydraulic Leakage Monitoring* (HLM) system is used to detect leakages and isolate defective sections of the hydraulic system by operating shut-off valves.

Hammarberg and Nadjm-Tehrani [36] report on the development of the electronic components of a HMS (white boxes in Figure 4), based on formal specifications written in the Esterel language. Esterel modules are used both for the system and the fault model, thus allowing verification of safety properties in the presence of faults. The main safety property is that no more than one valve be closed at the same time, since this condition could block the hydraulic system. The property has been verified by model checking combined with FTA and FMEA under several fault hypotheses. The Esterel model has then been automatically translated into VHDL, leading to the FPGA implementation. The authors compare the automatic generation of VHDL code from Esterel with handwritten and optimized code, and conclude that the overhead of the automatically generated code is acceptable for the application, given the resource availability afforded by the FPGAs.

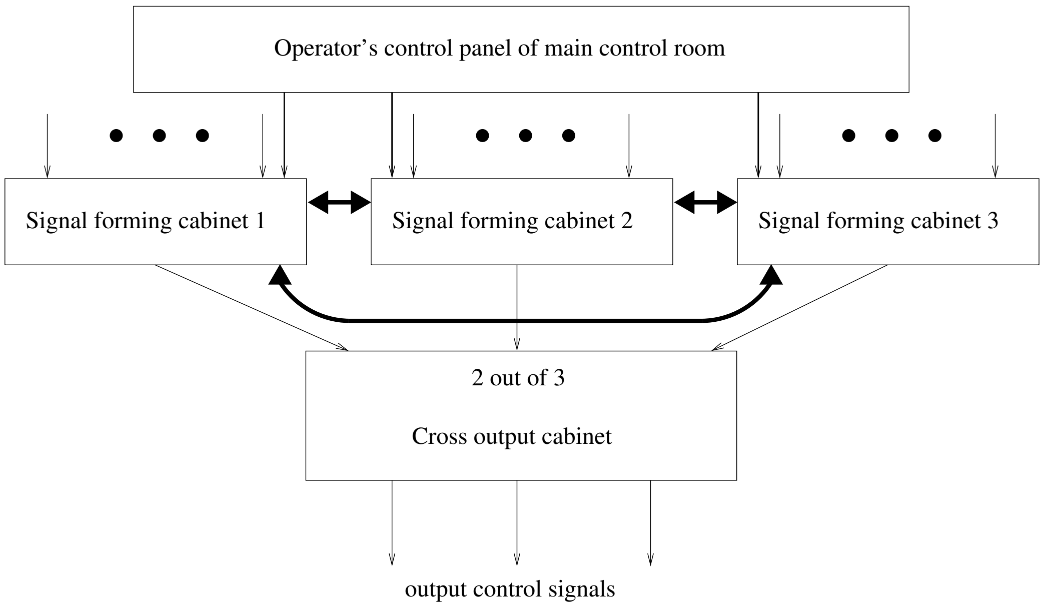

Figure 5: Reactor trip system (from [5]).

6.2 Reactor Trip System

Andrashov *et al.* [5] describe the development and V&V process used for the control logic of *reactor trip systems* (RTS) implemented with FPGA technology. The RTS is the central and most critical part of a nuclear powerplant's protection system. It samples sensor signals measuring physical magnitudes (e.g., temperature, pressure, or neutron flux), compares them with the allowed operating values, and shuts down the reactor if the prescribed limits are exceeded. Figure 5 shows the considered RTS, consisting of three signal channels feeding a two-out-of-three voter.

The adopted development process is an adaptation of the V-lifecycle, where the development branch of the V is divided in two main phases, *FPGA design* and *FPGA implementation*. The design phase consists in the *preliminary electronic design* subphase, where the system is modeled at the diagram level and verification is done by design review, and the *detailed electronic design* subphase, where system is modeled at the schematics and VHDL level, and verification is done by simulation and static analysis. The implementation phase consists in the subphases of *logic synthesis*, *placement and routing*,

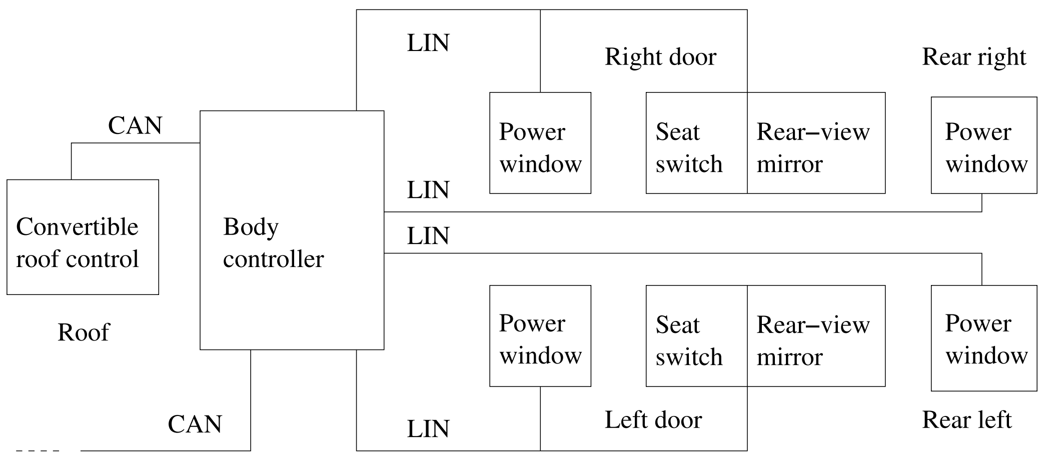

Figure 6: Body controller unit (from [62]).

and *bitstream generation*, where the system is modeled at the register transfer, netlist, and floorplan level, respectively. Verification is done by gate simulation and time simulation.

Special emphasis is placed on testing the RTS algorithms, i.e., the logic functions performed by the basic RTS subsystems. Thirty-four algorithms have been identified and tested by simulation with a 100% coverage of input value combinations chosen with the boundary value criterion.

6.3 Car Body Controller

Traub *et al.* [62] describe the development of an FPGA-based *body controller unit* (BCU) (Figure 6), in charge of controlling a car's electrically operated windows, rear-view mirrors, and other components. The considered BCU has been designed as a centralized FPGA controller replacing a number of electronic control units. The BCU includes a MicroBlaze processor, thus supporting the implementation of different functional parts of the application in software or in hardware.

The adopted development process is centered on model-based design, both for hardware and software. The BCU functions are modeled with Simulink and Stateflow diagrams, from which HDL code (for hardware modules) and C code (for software) is automatically generated. The code is then synthesized for the Xilinx Spartan 3 FPGA. The authors report data on resource requirement for different architectural approaches. An interesting results is that for some functional modules a hardware implementation may require more FPGA resources than a software implementation using the MicroBlaze processor.

7 Open Issues

Many issues are still unsolved and make the application of SRAM-based FPGA devices in the safety-related parts of systems still problematic. Three main points remain open. The first is the lack of methods and tools for the formal verification of netlists both at the logic and the post-place-and-route level: What designers can do is just simulate the obtained netlist trying to apply a sufficiently effective set of tests. This lack of methods has a twofold negative result: On the one hand it is very difficult to verify the functional equivalence of the synthesized netlist with respect of the original high-level behavioral specification, which forces designers to trust the correctness of the synthesis process implemented by the vendor tool; on the other hand it is unfeasible to verify the functional correctness and the absence of unwanted functionalities of third-party IP-cores provided at the netlist-level, which forces designers to rely on the correctness of the purchased netlist.

The second open point is the lack of open-source, freely modifiable and usable tools for the verification of the functional equivalence and absence of unwanted functionalities of the obtained bitstream with respect of the previously verified netlist. For example, the Change Detection Platform of Luna Inc. is a high-end tool developed in a research project of the US Defense Advanced Research Projects Agency, specifically addressed to the contractors of the US Department of Defense and not freely available. The lack of such tools, again, forces designers to rely on the correctness of the translation tool provided by the device vendor and on the trustworthiness of the IP-core provider.

Finally, partial dynamic reconfiguration in safety-critical applications represents a still open point. Modern devices largely support partial dynamic reconfiguration: This allows the system to both activate a given functionality only when needed and deactivate functionalities when not needed (thus saving space) and to adapt its behavior to changes in the working environment or in the user requirements [51]. On the other hand, the use of partial dynamic reconfiguration in safety-critical systems imposes hard requirements to the design, as discussed in [51]. In particular, the reliability of the data transfer is required if the new bitstreams are dynamically sent to the system when required, or a highly reliable persistent memory has to be used to store the unused bitstreams if the system is equipped with all the bitstreams since power-up. Furthermore, it must be ensured that the partial reconfiguration process does not affect the correct operation of the non reconfigured part, that may be required to keep working transparently.

An additional open issue is related to the assessment of the sensitivity to SEUs. Radiation testing may entail too high a cost for small-scale designs and fault emulation takes a very long time for real-world designs. This gets even worse when a number of iterations of design and sensitivity analysis is required before achieving an acceptably robust design. To alleviate these problems early assessment techniques have been proposed, in order to estimate the sensitivity of the system before implementing a prototype and thus allowing early corrections [59, 6, 38]. On the other hand, a number of techniques aimed at identifying the untestable faults in a design have been proposed [10, 12]. The identification of those faults that can never be tested allows reducing the number of faults that must be taken into account during the fault emulation process, thus speeding it up. On the other hand, what is still lacking is a unified framework of tools and techniques integrated in the standard design flow that may be used by designers at the various stages of the design.

8 Conclusions

This paper summarizes the design standards for the development of FPGA-based systems in safety critical applications together with the literature proposals, industrial and academic guidelines, and lessons learned from real projects.

Three main points about the design of FPGA-based systems in safety-critical application field can be identified. The first point is that it is strongly recommended to start the design of a safety-critical FPGA-based system only after a well structured and well documented design flow has been identified.

The second recommendation is never to trust completely the CAD tools provided by the FPGA device vendor, and always to verify the intermediate products of all phases of the design process using external tools (both simulation tools and formal methods).

Finally, even if the design and development process of an FPGA-based system is very much like the design and development process of a software system, the designer must know in depth all the technological details of the final target device that will host the system, such as special I/O pins, working frequency range, temperature, voltage and humidity ranges.

Tools and methodologies addressing these issues will boost the application of SRAM-based FPGA devices in safety-critical systems.

References

[1] Guideline for FPGA IV&V Demonstration Program, January 2007. National Aeronautics and Space Administration (NASA).

[2] Aarti Gupta, Ali Alphan Bayazit and Yogesh Mahajan. Verification Languages. In *Embedded Systems Handbook*, chapter 10, pages 301–321. Taylor & Francis Group, 2006.

[3] Miguel Angel Aguirre, Jonathan Noel Tombs, Vicente Baena, Fernando Muñoz, Antonio Jesus Torralba, A. Fernández-León, and F. Tortosa-López. Ft-Unshades: a New System for Seu Injection, Analysis and Diagnostics Over Post Synthesis Netlist. In *Proceedings of the 8th Military and Aerospace Programmable Logic Devices International Conference (MAPLD'05)*, 2005.

[4] M. Alderighi, F. Casini, S. D'Angelo, M. Mancini, S. Pastore, and G.R. Sechi. Evaluation of Single Event Upset Mitigation Schemes for SRAM based FPGAs using the FLIPPER Fault Injection Platform. In *Proceedings of the 22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT '07)*, pages 105 –113, sept. 2007.

[5] Anton Andrashov, Vyacheslav Kharchenko, Volodymir Sklyar, Alexander Siora, and Lubov Reva. Verification of FPGA-Based NPP I&C Systems: General Approach and Techniques. In *19th International Conference On Nuclear Engineering in Osaka (ICONE19 Osaka)*, 2011.

[6] Ghazanfar Asadi and Mehdi B. Tahoori. An analytical Approach for Soft Error Rate Estimation of SRAM-based FPGAs. In *Proceedings of the Military and Aerospace Applications of Programmable Logic Devices (MAPLD)*, pages 2991–2994, 2004.

[7] J. Ashley. Practical Guide to Certification and Re-Certification of AAvA Software Elements: Software for Programmable Logic Devices, July 2005. Praxis High Integrity Systems.

[8] R.C. Baumann. Radiation-induced Soft Errors in Advanced Semiconductor Technologies. *IEEE Transactions on Device and Materials Reliability*, 5(3):305 – 316, September 2005.

[9] C. Bernardeschi, L. Cassano, A Domenici, and L. Sterpone. ASSESS: A Simulator of Soft Errors in the Configuration Memory of SRAM-Based FPGAs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(9):1342–1355, Sept 2014.

[10] Cinzia Bernardeschi, Luca Cassano, and Andrea Domenici. SEU-X: a SEu Un-eXcitability prover for SRAM-FPGAs. In 18-*th IEEE International On-Line Testing Symposium (IOLTS)*, pages 29–34, 2012.

[11] Cinzia Bernardeschi, Luca Cassano, Andrea Domenici, and Luca Sterpone. Accurate Simulation of SEUs in the Configuration Memory of SRAM-based FPGAs. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT 2012)*, 2012.

[12] Cinzia Bernardeschi, Luca Cassano, Andrea Domenici, and Luca Sterpone. Unexcitability Analysis of SEUs Affecting the Routing Structure of SRAM-based FPGAs. In *Proceedings of the 23rd ACM International Conference on Great Lakes Symposium on VLSI*, GLSVLSI '13, pages 7–12, New York, NY, USA, 2013. ACM.

[13] Jonathan P. Bowen and Victoria Stavridou. Safety-critical systems, formal methods and standards. *Software Engineering Journal*, 8:189–209, 1993.

[14] W. Calienes Bartra and R. Reis. SET and SEU simulation toolkit for LabVIEW. In *12th European Conference on Radiation and Its Effects on Components and Systems (RADECS2011)*, pages 829 –836, sept. 2011.

[15] Francisco Cardells-Tormo, Javier Valls-Coquillat, Vicenc Almenar-Terre, and Vicente Torres-Carot. Efficient FPGA-based QPSK Demodulation Loops: Application to the DVB Standard. In *Proceedings of*

the Reconfigurable Computing Is Going Mainstream, 12th International Conference on Field-Programmable Logic and Applications (FPL '02), pages 102–111, London, UK, 2002. Springer-Verlag.

[16] Carl Carmichael, Earl Fuller, Phil Blain, and Michael Caffrey. SEU mitigation techniques for Virtex FPGAs in space applications. In *Proceeding of the Military and Aerospace Programmable Logic Devices International Conference (MAPLD)*, 1999.

[17] Carl Carmichael, Earl Fuller, Joe Fabula, and Fernanda D. Lima. Proton testing of SEU mitigation methods for the Virtex FPGA. In *Proceedings of the IEEE Microelectronics Reliability and Qualification Workshop*, pages 1–4, Pasadena, CA, December 2001.

[18] James A. Cercone, Michael A. Beims, and Kenneth G. McGill. Verification and Validation of Programmable Logic Devices. In *Proceedings of the 7th Military and Aerospace Programmable Logic Devices International Conference (MAPLD'04)*. National Aeronautics and Space Administration (NASA), September 2004.

[19] Henrik B. Christophersen, Wayne J. Pickell, Adrian A. Koller, Suresh K. Kannan, and Eric N. Johnson. Small Adaptive Flight Control Systems for UAVs using FPGA/DSP Technology. In *Proceedings of American Institute of Aeronautics and Astronautic (AIAA) Unmanned Unlimited Technical Conference, Workshop, and Exhibit*, 2004.

[20] Conmy, P. and Pygott, C. and Bate, I. A VHDL Guidance for Safe and Certifiable FPGA Design. In *Proceedings of the IET System Safety Conference*, October.

[21] R. Dobias and H. Kubatova. FPGA based design of the railway's interlocking equipments. In *Euromicro Symposium on Digital System Design (DSD 2004)*, pages 467–473, Aug.-3 Sept. 2004.

[22] A. Fernández-León. Field programmable gate arrays in space. *IEEE Instrumentation Measurement Magazine*, 6(4):42–48, 2003.

[23] Radio Technical Commission for Aeronautics (RTCA). DO-254 Design Assurance Guidance for Airborne Elecronic Hardware, April 2000.

[24] European Committee for Electrotechnical Standardization (CENELEC). EN 50128: Railway applications - Communications, signaling and processing systems - Software for railway control and protection systems, April 2002.

[25] European Committee for Electrotechnical Standardization (CENELEC). EN 50129: Railway applications - Communications, signaling and processing systems - Safety related electronic systems for signaling, February 2003.

[26] European Cooperation for Space Standardization (ECSS). Q-ST-60-02C Space product assurance: ASIC and FPGA development, July 2008.

[27] International Organization for Standardization (ISO). 26262-5: Road vehicles - Functional safety - Part 5. Product development: hardware level, December 2009. Draft.

[28] International Organization for Standardization (ISO). 26262-6: Road vehicles - Functional safety - Part 6. Product development: software level, December 2009. Draft.

[29] W. Gibbons and H. Ames. Use of FPGAs in Critical Space Flight Applications - A Hard Lesson. In *Proceedings of the Military and Aerospace Applications of the Programmable Devices and Technologies Conference*. National Aeronautics and Space Administration (NASA), 1999.

[30] M. Gokhale, P. Graham, E. Johnson, N. Rollins, and M. Wirthlin. Dynamic reconfiguration for management of radiation-induced faults in FPGAs. In *Proceedings. 18th International Parallel and Distributed Processing Symposium*, pages 28–38, April 2004.

[31] S. Golshan and E. Bozorgzadeh. Single-Event-Upset (SEU) Awareness in FPGA Routing. In *Proceedings of the 44th ACM/IEEE Design Automation Conference*, pages 330–333, 2007.

[32] J. Graf. Change Detection Platform for FPGA Trust. In *Proceedings of the Government Microcircuit Applications and Critical Technology Conference (GOMACTech-2011)*, March.

[33] P. Graham, M. Caffrey, J. Zimmerman, D. E. Johnson, P. Sundararajan, and C. Patterson. Consequences and Categories of SRAM FPGA

Configuration SEUs. In *Proceedings of the 6th Military and Aerospace Applications of Programmable Logic Devices (MAPLD'03)*, September 2003.

[34] Daniel González Gutiérrez. Single event upsets simulation tool functional description. Technical Report TEC-EDM/DGG-SST2, ESA-ESTEC, 2000.

[35] Sandi Habinc. Lessons Learned from FPGA Developments, September 2002. Technical Report. Gaisler Research.

[36] Jerker Hammarberg and Simin Nadjm-Tehrani. Formal Verification of Fault Tolerance in Safety-Critical Reconfigurable Modules. *International Journal on Software Tools for Technol Transfer*, 7(3):268–279, 2005.

[37] J. Heiner, N. Collins, and M. Wirthlin. Fault Tolerant ICAP Controller for High-Reliable Internal Scrubbing. In *IEEE Aerospace Conference*, pages 1–10, March 2008.

[38] O. Heron, T. Arnaout, and H.-J. Wunderlich. On the reliability evaluation of SRAM-based FPGA designs. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL'05)*, pages 403–408, August 2005.

[39] Adrian J. Hilton, Gemma Townson, and Jon G. Hall. FPGAs in Critical Hardware/Software Systems. In *Proceedings of the 2003 ACM/SIGDA eleventh international symposium on Field programmable gate arrays (FPGA '03)*, pages 244–244, New York, NY, USA, 2003. ACM.

[40] W.K. Huang, F.J. Meyer, N. Park, and F. Lombardi. Testing Memory Modules in SRAM-based Configurable FPGAs. In *Proceedings of the International Workshop on Memory Technology, Design and Testing*, pages 79 –86, aug 1997.

[41] IBM. SOI Technology: IBM's Next Advance in Chip Design, January.

[42] International Electrotechnical Commission (IEC). 61508-2: Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 2. Requirements for electrical/electronic/programmable electronic safety-related systems, May 2000.

[43] International Electrotechnical Commission (IEC). 61508-3: Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 3. Software requirements, May 2000.

[44] Fernanda Lima Kastensmidt, Luigi Carro, and Ricardo Reis. *Fault-Tolerance Techniques for SRAM-Based FPGAs (Frontiers in Electronic Testing)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[45] Ian Kuon and Jonathan Rose. Measuring the gap between fpgas and asics. *IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS*, 26(2), February 2007.

[46] Ian Kuon, Russell Tessier, and Jonathan Rose. Fpga architecture: Survey and challenges. *Foundations and Trends in Electronic Design Automation*, 2(2):135–253, 2008.

[47] A. Ledeczi, P. Volgyesi, M. Maroti, G. Simon, G. Balogh, A. Nadas, B. Kusy, S. Dora, and G. Pap. Multiple simultaneous acoustic source localization in urban terrain. In *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, pages 491–496, April 2005.

[48] Luís Gomes, João Paulo Barros and Anikó Costa. Modelling Formalisms for Embedded System. In *Embedded Systems Handbook*, chapter 5, pages 134–168. Taylor & Francis Group, 2006.

[49] A. Mazzeo, L. Romano, G. P. Saggese, and N. Mazzocca. Fpga-based implementation of a serial rsa processor. In *DATE '03: Proceedings of the conference on Design, Automation and Test in Europe*, page 10582, Washington, DC, USA, 2003. IEEE Computer Society.

[50] Eugene Normand. Single event effects in avionics and on the ground. *International Journal of High Speed Electronics and Systems*, 14(02):285–298.

[51] B. Osterloh, H. Michalik, S.A. Habinc, and B. Fiethe. Dynamic partial reconfiguration in space applications. In *Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems*, pages 336–343, July 2009.

[52] S. Rezgui, J. J. Wang, Y. Sun, B. Cronquist, and J. McCollum. Configuration and Routing Effects on the SET Propagation in Flash-Based FPGAs. *IEEE Transactions on Nuclear Science*, 55(6):3328–3335, Dec 2008.

[53] Davide Sabena, Luca Sterpone, M. Schölzel, T. Koal, H.T. Vierhaus, S. Wong, R. Glein, F. Rittner, C. Stender, M. Porrmann, and J. Hagemeyer. Reconfigurable high performance architectures: How much are they ready for safety-critical applications. In *Proceedings of 19th IEEE European Test Symposium (ETS)*, pages 175–182, 2014.

[54] S. Schulz, G. Beltrame, and D. Merodio-Codinachs. Smart behavioral netlist simulation for SEU protection verification. In *9th European Conference on Radiation and Its Effects on Components and Systems (RADECS2008)*, pages 406–411, sept. 2008.

[55] J. She and J. Jiang. Application of FPGA to Shutdown System No.1 in Candu. In *Proceedings of the 6th American Nuclear Society International Topical Meeting on Nuclear Plant Instrumentation, Control, and Human-Machine Interface Technologies (NPIC&HMIT 2009)*. American Nuclear Society, 2009.

[56] Andreas Söderberg, Jacques Hrard, and Lars Bo Mortensen. Guideline for Design and Safety Validation of Safety-Critical Functions Realized with Hardware Description Language, May 2005. Technical Report. NORDTEST: a Nordic Innovation Center Brand.

[57] L. Sterpone and N. Battezzati. A new placement algorithm for the mitigation of multiple cell upsets in SRAM-based FPGAs. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '10, pages 1231–1236, 3001 Leuven, Belgium, Belgium, 2010. European Design and Automation Association.

[58] L. Sterpone, M. Sonza-Reorda, and M. Violante. RoRA: a reliability-oriented place and route algorithm for SRAM-based FPGAs. In *Research in Microelectronics and Electronics, 2005 PhD*, volume 1, pages 173–176, July 2005.

[59] L. Sterpone and M. Violante. A new analytical approach to estimate the effects of SEUs in TMR architectures implemented through SRAM-

based FPGAs. *IEEE Transactions on Nuclear Science*, 52(6):2217–2223, December 2005.

[60] Luca Sterpone, Mario Porrmann, and Jens Hagemeyer. A novel fault tolerant and runtime reconfigurable platform for satellite payload processing. *IEEE Transactions on Computers*, 62(8):1508–1525, 2013.

[61] Angela Sutton. No Room for Error: Creating Highly Reliable, High-Availability FPGA Designs, November. Synopsis Inc.

[62] Matthias Traub, Oliver Sander, Arik Rathner, and Jürgen Becker. Generating Hardware Descriptions from Automotive Function Models for an FPGA-Based Body Controller: A Case Study. In *MathWorks Automotive Conference '08*, 2008.

[63] J. J. Wang, Brian Cronquist, John McCollum, Frank Hawley, Donald Yu, Richard Chan, Rich Katz, and Igor Kleyner. Total dose and SEE of metal-to-metal antifuse FPGA. In *Proceedings of the 2nd Military and Aerospace Applications of Programmable Devices and Technologies Conference*, volume 80, pages 1–6, 1999.

[64] H.-R. Zarandi, S.-G. Miremadi, D.K. Pradhan, and J. Mathew. SEU-Mitigation Placement and Routing Algorithms and Their Impact in SRAM-Based FPGAs. In *Proceedings of the 8th International Symposium on Quality Electronic Design*, pages 380–385, 2007.