Fault Handling in FPGAs and Microcontrollers in Safety-Critical Embedded Applications: A Comparative Survey

Falk Salewski
Embedded Software Laboratory
RWTH Aachen University, Germany
salewski@informatik.rwth-aachen.de

Adam Taylor
Selex Sensors and Airborne Systems
Basildon, SS14 3EL, UK
Adam.Taylor@selex-sas.com

***Abstract*— Safety-critical systems nowadays include more and more embedded computer systems, based on different hardware platforms. These hardware platforms, reaching from microcontrollers to programmable logic devices, lead to fundamental differences in design. Major differences result from different hardware architectures and their robustness and reliability as well as from differences in the corresponding software design. This paper gives an overview on how these hardware platforms differ with respect to fault handling possibilities as fault avoidance and fault tolerance and the resulting influence on the safety of the overall system.**

I. Introduction

According to the growing complexity and a rising need for flexibility, nowadays safety-critical systems[1] include more and more embedded computer systems. These computer systems could be realized using many different hardware platforms ranging from CPU based systems as microcontrollers (MCU) and digital signal processors (DSP) to Programmable Logic Devices (PLD)[2] as Complex Programmable Logic Devices (CPLD) and Field Programmable Logic Arrays (FPGA). Additionally, each type of hardware platform can be based on different *device techniques* (silicon structure, memory type).

In this paper, we give a comprehensive comparison of FPGAs and MCUs, as popular representatives of CPU and PLD based embedded systems, with respect to their suitability for safety-critical applications. Although a certain functionality could be realized on both types of hardware platforms in principle [40] and both are programmed on a higher abstraction level (MCUs typically in C or ADA, FPGAs typically in hardware description languages (HDL) as VHDL or Verilog), these different platforms lead to fundamental differences in design.

In case of MCUs, the hardware is determined at fabrication. The hardware consists of a central processing unit (CPU), a program memory and a data memory. There is always an explicit separation between the data path and the processing unit. All commands are read from the program memory and executed sequentially. MCUs contain interrupt processing allowing reactive concurrent behavior. Additionally, several special purpose hardware modules, so called *on-chip peripherals* as timers and communication interfaces are integrated.

In case of FPGAs, the hardware is programmable. A high number of programmable logic blocks is available, which can be individually connected by programmable interconnect matrices. Additionally, programmable I/O blocks are available for interfacing (see e.g. [36]). The existence of a separation between the data path and the control logic depends on the architecture chosen by the designer. Execution is of parallel nature, but sequential behavior can be realised through the use of state machines, control / enable signals and pipelining.

When considering safety and reliability, major differences result from different hardware architectures and their robustness and reliability as well as from the corresponding different software designs. However, just a few of these differences are considered during the hardware selection process, mostly focusing on the robustness of the hardware platforms only. In our opinion, all factors influencing aspects of safety-critical systems should be considered during hardware selection.

The remainder of this paper is organized as follows. In section II, issues affecting the design of safety-critical systems are presented while in the following section III different aspects of fault handling are examined for MCUs and FPGAs. Finally, a conclusion is given in section IV.

Note: All physical parts of an embedded system are referred to as hardware (e.g. MCU, FPGA), while the languages which determine the behavior of these systems are referred to as software (e.g. assembly, C, VHDL). It has to be noted, that processors (CPUs) could also be integrated into FPGAs. However, these so called *soft-processors* and their specific impacts could not be considered in this paper explicitly.

II. Design of Safety Critical Systems

The traditional approach for designing safety-critical systems focuses on functional requirements of the considered system in the first step. Although safety requirements are of great importance, this property is typically checked comparatively late in the design cycle. In many instances the testing of safety requirements will only occur after the functionality has been verified and if changes are necessary at this stage to fulfill the safety requirements the cost to completion will rise

[1] A system is safety-critical, if a failure of the system could lead to consequences that are determined to be unacceptable [21].

[2] PLDs are also known as *reconfigurable hardware*

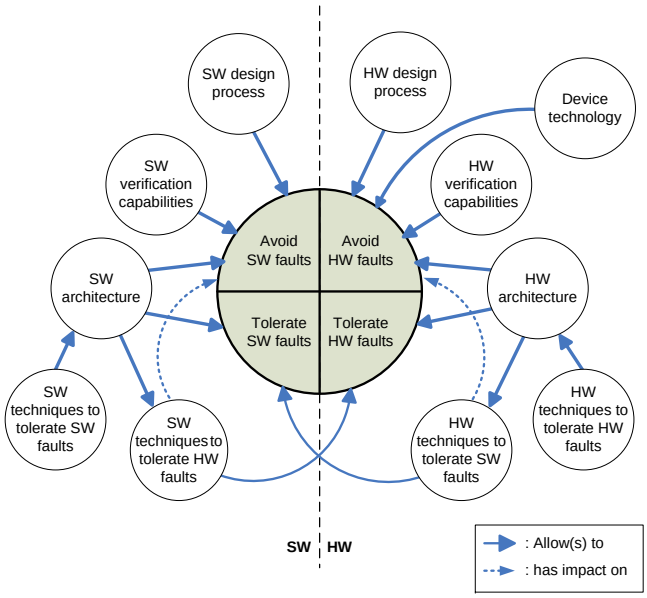

Fig. 1. Fault handling in embedded systems: software vs. hardware issues

dramatically. Accordingly, it would be beneficial to know in advance how certain design decisions would affect the safety of the resulting system. This demands suitable rules regarding design decisions influencing reliability.

For this reason, the influences of design decisions on the safety and reliability of a system have to be identified. These influences include impacts on software, hardware and system issues and a first collection of some of these impacts can be found in [37]. A closer analysis of these impacts revealed that they all target different types of fault[3] handling techniques. During design time, measures can be applied to *avoid* that faults occur in the later system as for example testing or formal verification. Other measures try to *tolerate* faults at run time which have not been detected and removed during design time. Faults can occur as well in hardware as in software and fault handling techniques are also possible in hardware and/or software. Our analysis revealed several aspects related to design decisions which influence these different fault handling techniques. To organize these aspects we propose a structured representation as depicted in Fig. 1. This figure consists of two major parts: fault handling in software (left part) and fault handling in hardware (right part). Both parts could be influenced by the design process, available verification capabilities and the architecture chosen. Additionally, a factor influencing the fault handling in hardware is the device technology used for the actual device. Several fault handling techniques are known for software and hardware and several of these aspects depend on the hardware platform chosen.

In the following sections we will discuss these hardware dependent safety aspects for embedded systems based on MCUs and FPGAs. It has to be mentioned that these embedded systems have special properties which complicate the design process: First of all, they are very often (hard) real-time systems. In this case, the correctness of operation depends upon both the correctness of the output and the time at which it is calculated. All measures introduced into the design to avoid and tolerate faults must be able to fulfill these real-time requirements. Secondly, embedded systems typically have strong HW/SW interactions. Accordingly, when talking about reliability and safety, an **overall view** ranging from elementary hardware issues up to system issues has to be applied.

[3]In accordance with [2] we define *fault* in the following context: *The manifestation of a fault will produce errors in the state of the system, which could lead to a failure.*

III. Fault handling: MCU vs. FPGA

As already mentioned, fault handling can be applied to handle faults in hardware and in software (Fig. 1). In the next subsection we will have a look at the handling of hardware faults followed by a subsection dealing with the handling of software faults. At the end of this section, we will discuss system issues affecting both, hardware and software.

A. Handling of Hardware Faults

This section should give an brief comparative overview on the handling of hardware faults in MCU and FPGA based embedded systems (aspects listed in the right part of Fig. 1).

1) Hardware Design Process: The hardware design process has a high impact on the quality of the resulting hardware. Especially measures to obtain certain levels of quality (e.g. testing, burn in) are of great importance for the avoidance of faults in the later device. However, these impacts of the hardware design process are probably very similar for MCUs and FPGAs and are not considered any further for this reason.

2) Device Technology: Publications dealing with hardware reliability often target specific problems in aerospace or space applications (e.g: SEU (Single Event Upset), SEL (Single Event Latch-up), TID (Total Ionizing Dose)). These effects are less critical at ground level, but still exist [4], [6], [30]. Especially [30] gives a good overview how memory cells could be affected at ground level. These effects have to be considered in safety-critical systems since a fault in the hardware of such a system could lead to catastrophic consequences. Measures have been introduced to harden the silicon structure. For example, alpha particles can be shielded by a thick polyimide layer prior to package. However, this does not work for high energy cosmic radiation [4]. TID ratings can be improved through special circuit design techniques and shielding methods [44], and SEL has been all but eliminated for many FPGA devices through the incorporation of special current limiting circuitry [44]. However, devices including these measures are targeted primarily at space based applications and as such their cost is likely to be prohibitive for use in many designs. Furthermore, SEU cannot be avoided completely by device techniques and has to be dealt with in any CMOS device.

One question is, whether some device techniques are suited better than others to built reliable devices. For example, ”old” silicon techniques could be more robust since increasing device density and larger dies of current devices makes these less reliable [22]. However, there have been improvements in

silicon techniques as can be seen in [4]. Some sources of soft errors were detected in the last decades and according measures to avoid these are present in newer techniques only (e.g. removal of borophosphosilicate glass (BPSG) to avoid soft errors from low energy cosmic neutron interactions [4] or integration of shielding). The use of this improved techniques in combination with a large scale process to increase the robustness of the circuitry might be an option theoretically, but this concept is not applicable, since safety-critical applications currently do not drive the chip market but non-critical applications as consumer electronics which focus on low cost and high performance. The use of older devices could lead to the problems mentioned before. Furthermore, designing in older devices can lead to a major problem in obtaining devices for repairs or new builds of the system especially on many large safety-critical systems which have long (15-25 years) operating periods.

The next aspect often discussed is the memory technique used for program and configuration memory. One time programmable (OTP) memories, as for example anti fuse technology, are very robust to soft errors [25]. However, they are less flexible and it has to be kept in mind that even devices, based on anti fuse program/configuration memories include SRAM for storage of dynamic data. In-circuit reconfigurable memories allow much higher flexibility than OTP devices. In [44] they state that the most promising in-circuit reconfigurable devices from the TID, SEL, and SEU point of view are those that are based upon FLASH or EEPROM. The most flexible but also most sensitive type are SRAM based memories. As already stated above, all devices include SRAM for storage of dynamic data. Accordingly, soft error mitigation techniques have to be applied on all devices and the only difference is the effort that has to be taken to check the program/configuration memory. Important factors here are how fast an error can be detected and how long it takes to mitigate the error.

The authors of [20] indicate that (in case of FPGAs) it is not only the feature size, but also the supply voltages and the architecture itself which influence the hardware reliability. Furthermore, it is obvious that larger amounts of memory lead to larger arrays of sensitive chip areas leading to higher probabilities of soft errors. However, fault mitigation can be applied comparatively easy in case of program/configuration memories (see Sec. III-A.4). Other possible impacts are temperature (own and external), humidity, altitude and mechanical shock which have to be considered for all CMOS devices.

3) Hardware Architecture (Fault Avoidance): Certain hardware architectures allow techniques to avoid and to tolerate hardware faults. In this section, architectural measures to avoid random and systematic hardware faults are looked at, while fault tolerance techniques for random hardware faults are discussed in the following section.

The hardware architecture requires considerable thought as it will be the hardest to modify later in the design cycle. Standard hardware considerations should apply e.g. signal integrity, power supply decoupling and bypassing, input protection and component derating. Aspects of the design which may not normally require consideration may also need consideration, as for example the implementation of communication buses between devices. Both, parallel and serial buses each have advantages and disadvantages to their use. Serial buses will allow the *stuck at faults* (signal is permanently high or low) to be identified easier however, they may lead to a reduction of the data throughput of the system.

The architecture can also reduce the SEU sensitivity as mentioned in the previous section. MCUs and FPGAs are affected differently by transient hardware failures. In case of MCUs, the program memory, the data memory and general and special purpose registers could be corrupted. In case of FPGAs the configuration memory, FlipFlops and, if available, the data memory could be corrupted. Changes in the configuration memory could fundamentally change the behavior of the FPGA. However, the same is true, if certain registers are affected in MCUs [42].

One area of the design that requires careful design and analysis in both FPGA and MCU based systems is often one of the most easily over looked areas and that is the Joint Test Action Group (JTAG) interface (IEEE standard 1149.1) [20]. The JTAG interface is used in digital electronics to program FPGAs, Flash etc, and can also be used for debugging of both FPGAs and MCUs and is used increasingly in production testing to ensure the board has been manufactured correctly. This interface can be invaluable during the design, development and production testing however, it is important to ensure that this interface remains inoperable during normal operation. If the Test Access Port (TAP) control state machine is not correctly disabled then a test mode could be incorrectly entered during normal operation due to noise at the JTAG pins appearing as a valid command to the TAP state machine and could result in unpredictable operation of inputs and outputs [27]

All these factors should be considered during hardware platform selection. However, no general differences in hardware fault avoidance in the architectures of FPGAs and MCUs could be found. Next we look at measures to tolerate hardware faults by hardware and software measures.

4) Hardware & Software Architecture (Fault Tolerance): The tolerance of hardware faults can be done in hardware and/or in software (see Fig. 1). We look at transient faults in this section while permanent faults are considered in section III-C.

If we look at the different hardware platforms we can identify different system components which can be affected by transient hardware faults. In case of the MCU these are the program memory, the data memory and general and special purpose registers, in case of the FPGA these are the configuration memory, the data memory and FlipFlops.

MCU: If an upset occurs in the program or data memory of an MCU, Error Correction Codes (ECC) can be used to mitigate the error. This can be done in software (no specific hardware needed, memory and computation time overhead, might increase complexity of the software), but several actual MCUs are available with hardware ECC for program and/or data memories also (increased hardware costs, no software

overhead). Mitigation of faults in CPU registers could be realized in software with time redundancy (redoing calculations and compare results). The protection of MCU registers as program and stack counter is more complicated and a pure software approach leads to high overhead (e.g. speed decrease (3 times) and memory overhead (4.5 times) [28]) and most probably complicates the software architecture and the verification of real-time requirements.

An approach using standard hardware, namely to use superscalar processors and duplicate instructions, has been proposed in [31] for mitigation of transient hardware faults. Other approaches propose to integrate specific hardware based reliability measures in processor based systems, as in [41] (making registers SEU tolerant), in [6] (control flow checking) and in [7] (data checking). Another approach was an MCU for automotive applications which included BIST (Built In Self Test) [3], however it is not known if it ever found its way into a real application. But these days more and more MCUs are available which at least include built in ECC for data and program memory as already mentioned above. Other devices include further reliability measures. The ARM CORTEX processor targeted at SoC designs has been designed with automotive safety-critical applications in mind. One of the features offered is the ability at synthesis time to duplicate the processor logic and implement logic to check if both sets of logic agree (lock step). Through the processors BIST interface all of the RAM blocks within the processor can be accessed and tested. Support for byte-wise parity bits and ECC are also provided however, ECC logic must be implemented by the hardware design team [32].

A similar approach to the duplicate CORTEX processors and error checking maybe undertaken utilising the embedded Power PCs available within some members of the Xilinx Virtex FPGA series of devices. These processors can be made to function in lock step with both processors performing the same calculations from the same data, the design team can then implement error detection logic within the FPGA which traps errors should a difference occur between the two [46]. An approach with a heterogeneous dual core device in combination with additional hardware measures as hardware CRC and memory protection units based on an INFINEON TC1766 can be found in [9]. Further approaches as the use of co-processors, available in several MCU devices, for fault mitigation are known but usually these co-processors can monitor only specific functionalities of the main processor.

FPGA: While one major advantage of FPGAs with respect to reliability is that the integration of hardware reliability measures is possible on **generic** hardware, FPGAs have the disadvantage, that soft errors could not affect only their memories used during processing but the hardware structure itself. However, the configuration of an FPGA can be checked in different ways, depending on the FPGA device used. One method recommended is using an inbuilt constantly running CRC check running in the background of the FPGA to determine configuration errors and allow a reconfiguration if an error is detected [1]. Another option suggested is to read back the configuration data from a configured FPGA and to check the received CRC against the known good CRC, allowing reconfiguration if there is a mismatch [47]. Both of these approaches require additional hardware. In the case of the first approach, additional circuitry is required to detect faults and reconfigure the FPGA while in the second approach a configuration controller is required to configure the device and then initiate the read back CRC checks. Further on, the addition of such a CRC can affect the design. In a simple case study, conducted for this paper, a sample design which utilized a Phase Lock Loop (PLL) and a series of counters was implemented on an Altera Cyclone target hardware, both, with and without the CRC check. The implemented design without a CRC check was capable of operating at 169MHz while enabling the CRC check (implemented through the use of the software switch provided within the place and root tool) reduced the maximum operating frequency to 153 MHz. The time taken to fit the design within the selected device increased by approximately 10%. The read back of the configuration data may also be used for flash based FPGA technologies, however a read back scheme means that the device contents cannot be locked to prevent reverse engineering.

In [22] an approach of dynamic fine grained reconfiguration is proposed to overcome the problem of configuration faults. In [43] a self configuration checker was developed while this obviously requires considerable effort to implement, it is a very elegant solution to the problem. The self checker was developed using a Xilinx virtex II Pro FPGA with suggestions for partitioning into a radiation hard FPGA should a higher level of robustness be required. Another option is TMR (Triple Modular Redundancy) which needs more chip area (3x original hardware + voter) while partial TMR [33] is an option to reduce the area overhead in some applications. In [35] it has been proposed to use alternating logic to detect hardware faults. Success of TMR and other FPGA reliability measures can be validated with tools as presented in [33], [45].

The vulnerability of the memory holding the original configuration file has to be considered also [14]. However, standard error detection and correction measures, as those presented in the MCU section above, could be applied. Additionally, error detection within the configuration memory can be achieved if system tests are performed at start up confirming that the FPGA is correctly configured.

FPGA's provide significant choice to the safety-critical system designer, allowing the integration of safety measures / redundancy in either the same device or implementation across multiple devices as mentioned above. Which approach is followed depends upon many different factors. These are application specific however, common factors will be the safety level required, the failure rate (FIT) of the device selected, and the size of the logic design (is it possible to fit more than one implementation of the design within the device selected). The physical size of the printed circuit board may also act as a constraint, as it may not be possible to provide the space required to implement three separate FPGAs and the required voter. The available power budget may also limit

the design choice as FPGAs can be power hungry devices especially during the power up sequence.

Care must also be taken when designing the safety-critical FPGA system to ensure that single points of failure do not lead to the system failing. Single points of failure are most likely to occur within a single FPGA implementing internal TMR. In this situation careful thought must be given to the handling of reset and clocks especially if a single crystal is to be used. Utilising single input pins for signals which are then fed to different instantiations of the design is not recommended as a failure of the input cell either on the circuit card or internal to the FPGA input cell would result in a failure which might be difficult to detect. A better approach is to provide an input pin for each of the signals required by the different instantiations as this would enable a single failure of an input cell to be detected. A similar approach can be taken with clocks and resets using different global input pins for the clock and reset for each of the instantiations reducing the ability of a single point of failure.

Finally, a brief discussion of safety-related differences between FPGAs and MCUs can be found in [12] and according to [44] FPGAs will only serve to enhance the fault tolerant design options.

5) Hardware Verification: Another important aspect in safety-critical systems is to show that the hardware will always behave as intended. One approach is the formal verification of the hardware built (with respect to specification) which is applicable for both MCU and FPGA in the same way.

If devices are taken which have been used in other applications without failures for a certain time, they can be considered as ”proven in use” which is sufficient for less critical systems [18]. However, this implies not to use new state of the art technologies which could lead to the problems aforementioned. It might be interesting to analyse whether FPGA or MCU hardware have the higher risk of unrevealed hardware design faults after being used in a number of applications. If new hardware is to be qualified (temperature, humidity, altitude, radiation, shock vibration, acceleration etc.) a single type of FPGA can be taken which can be used across a high number of applications later on (generic hardware, provided it is of a reasonable size) while it is unlikely that a single MCU can be used across many different applications due to specialization (fixed/floating point, on chip peripherals, etc.).

B. Handling of Software Faults

Increasing complexity, real-time requirements and hardware/software dependencies make developing reliable software for embedded systems a non trivial task. Additionally, software measures for hardware fault mitigation, as described in the previous section, further increase the software complexity. Handling these challenges involves several aspects (see left part of Fig. 1) which will be looked at in the following.

1) Design Process: The software design processes of MCUs and FPGAs are different, especially from the programming language point of view. The two main programming languages used for MCUs today are C/C++ and ADA and there has been much work conducted on which is more suitable for safety-critical applications. Many governmental and international organisations which develop safety-critical systems and safety standards as [18] prefer ADA rather than C. However, the automotive industry has developed a subset of the C language called MISRA C which has been designed to address C’s failings in the safety-critical arena [26].

Although currently most FPGAs are programmed in VHDL and Verilog, recent publications propose the use of languages developed for CPU based systems as ADA [16] or Esterel [15], [17]. They argue, that classical HDLs are not suited for more complex designs and that languages as Esterel allow formal verification. However, it is important that PLD’s can not be treated as CPU based systems, especially regarding the HDL used to implement the design. While the HDL may look like a software programme it is not, instead it describes how the implemented hardware is to function. Unlike a software programme which will execute sequentially, the implemented FPGA will be operating simultaneously. Therefore the HDL development process will be subject to different rules and procedures than the software development to ensure fault tolerance or avoidance. Further on, additional transformations (e.g. Esterel to VHDL) are needed if ”MCU languages” will be applied which have to be verified. And it has to be noted that formal verification is also possible for systems written in VHDL [8], [11], [29] and is applied in industry already.

Motivations for higher programming languages are important to handle the increasing complexity of nowadays embedded systems. However, in embedded software design the underlying hardware has to be considered in safety-critical applications, since many faults appear at this hardware/software boundary (especially timing problems). This consideration has to be done manually or by suitable tools and might be more challenging for FPGA than for MCU designs.

The FPGA design process of specifying, designing and verifying individual modules probably allows for better reuse of HDL components within the FPGA through the use of generics which can be used to modify vector widths or simple module options. The reuse of common components across the system reduces the possibility of errors being introduced by different design engineers implementing the same function, this approach can also lead to a reduced time to market. This library of components can be either written at the start of a project or built up over a series of projects.

A similar advantage using FPGAs is the possibility of using Commercial Off The Shelf (COTS) components provided by the third parties (in VHDL or Verilog). These COTS components are available from both the device manufacturers and third party specialists, and due to the similar architecture of FPGA’s these third party components are normally suitable for use across different FPGA families. The use of these modules will further reduce development time, provided they are subjected to an in depth test bench which checks the accuracy, time response and behaviour during failure modes. COTS components also have a much wider user base across

many different applications. This wider user base and the varied applications of the components will inevitably mean that errors within the component design are identified quicker and corrected by the manufacturer.

2) Software Architecture: All faults in software are systematic. In other words, they are introduced into the system at design time. Accordingly, any approach of software fault avoidance has to be integrated in the software design process. However, if the choosen hardware platform fits well in the application this might ease the design. FPGA based solutions can offer a benefit in the design, timing, verification and design reuse when required to perform mathematical calculations / transfer functions. The safety-critical design process allows the generation and verification of a series of reusable mathematical modules (multiplier, divisors, square root, linear interpolators). Using VHDL generics the vector widths of both the input data and result can be modified to the width required and used across multiple FPGA designs. This set of common components can then be instantiated with new HDL module which implements the required transfer function. The transfer function can then be verified through the use of an test bench which tests the module meets the required time response, accuracy and failure mode behaviour. If the system design requires, these modules can then be easily duplicated or triplicated when instantiated within the hierarchical HDL to create a 1oo2 (one **o**ut **o**f two) or 1oo3 implementation. This approach works well when the transfer function will not change during run time for example required constants are loaded into the module at start up (offsets, sensors characteristics). If it is required that the module result is calculated using different transfer functions depending upon run time options a MCU will be more suitable as this will typically lead to the development of an Arthimetic Logic Unit (ALU). In case of an FPGA this can lead to problems ensuring the timing and accuracy are achieved for each possible transfer function as most mathematics will be implemented using a fixed point number system and many of the inputs will have the radix point located at different positions within the input. An MCU based approach, especially if it is capable of floating point operations, would be capable of meeting the timing and accuracy with greater ease than an FPGA in this case.

In case of MCUs, several techniques are known to handle SW faults as information hiding, recovery blocks or defensive programming in general (see e.g. [18]). However, it has to be considered that some of these measures influence the timing of the device which is especially critical in real-time applications.

Software architecture plays a major role in FPGAs since it allows the integration of effective reliability measures not available in todays CPU based systems, although trends to include them in MCUs are present (cf. section III-A.4). In FPGAs, functionality can be spread in different encapsulated blocks and does not have to be executed by a single CPU which offers possibilities to reduce the system complexity (if the different blocks are not strongly coupled). Safety measures (dealing with hardware and software faults) could be integrated in FPGAs without influencing the main functionality for the same reason which is especially interesting for hard real-time tasks.

3) Software Verification: According to [19], verification has become the bottle neck of the design process. Verification is especially important for safety-critical systems [24], [39], and it is desirable to achieve very high test coverage (100% is desired) on all of the required metrics for this reason. While this does add to the development timescales, work done by [13] indicates that an increase in the code coverage is likely to increase reliability. While this work was conducted on traditional software there is no reason to believe the results could not be extended to HDL code coverage (in [19] an introduction to different coverage metrics used in HDL design can be found).

FPGAs offer a benefit over a more traditional MCU based approach in that execution is more deterministic on a PLD than on an embedded microprocessor using techniques as interrupt processing and cache memory [44]. This eases verification of critical real-time systems and the fact that a frequently encountered source of failures in hard real-time systems is the inability of the system to deliver the required services by the required deadline under various workload conditions [23] makes this an very important aspect.

Techniques like assertion based verification and code coverage analysis demand modifications of the code. In case of CPU based systems, this changes the timing properties which is problematic for real-time systems. In FPGAs, assertions usually do not affect the timing properties of the application.

Formal verification of the software developed is another important issue. Approaches are available for systems written for FPGAs [8], [11], [29] and for MCUs [38] used in embedded systems. However, formal verification of real-time properties remains a challenge for model checking of MCU code.

Formal Equivalence Checking used in FPGA designs verifies that the *synthesis* and *place and route* stages (which typically occur after functional verification has been completed) have not introduced or removed logic therefore changing the function of the logic. This verification is needed, since design tools will attempt to modify and optimise the logical structures described within the HDL to make it best fit the logical structures available within the target architecture. It is therefore important that in a device intended for safety-critical applications these downstream processes do not change the behaviour of the module under error or error free conditions. Formal Equivalence Checking provides the ability to check that the functionality described in a HDL (typically Register Transfer Level) is the same as that of the implemented netlist.

Figure 2 from an example located within [34] illustrates one of the primary reasons why formal equivalence checking is so important. While the original design was incapable of asserting both Q and !Q at the same time even if a SEU occurred, the implementation would allow both Q and !Q to be asserted if a SEU were to occur possibly resulting in undefined behaviour of downstream modules.

Utilising formal equivalence checking does in the authors experience require the disabling of many of the synthesis

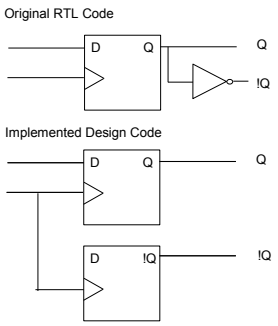

Fig. 2. The effects of enabling replication in the downstream flow

tools optimisation options. The author found equivalence could be achieved within a reasonable timescale through the disabling of pipelining, retiming, replication, not gate push back, resource sharing and hard limiting the fan out. This therefore results in the design engineer implementing many of the optimisations within RTL as opposed to allowing the synthesis tool free reign, if high clock frequencies are to be utilised. State machines and counters must also be carefully designed to ensure that any unused states are addressed and do not lead to differing behaviour between the RTL and the final netlist. It is often best to ensure all state machines are encoded using a sequential or grey code, as one hot implementation can often require transformations or constraints within the Formal Equivalence tool which reduce the confidence that all registers and all their possible values have been checked. The use of synthesis / place and route settings to implement safe state machines will lead to the addition of hardware within the final netlist which is not present within the Golden RTL and will therefore result in a Formal Equivalence Checking mismatch. Work done by the NASA Office of Logic Design in [5] and [10] corresponds with the authors experience with regard to synthesis and place and route optimisations. The state machine implementation technique described within [10] is very interesting and deserve further investigation with regard to the ease of formal equivalence checking.

C. System Safety & Reliability Issues

Safety is a system problem [24] and it has to made sure, that the combination of hardware and software never brings the system into an unsafe state.

One problem is that validating safety-critical systems completely overwhelms the cost of their design and construction and software validation is a major component of this cost [23] (referring to MCUs). In order to ease verification, Lala and Harper proposed in [23] to partition redundant elements into individual fault-containment regions. These should be provided with independent power and clocking sources and their interfaces should be electrically isolated. As already discussed before, partition of redundant elements could not be done in a single MCU, but to some instance on a single FPGA (electrical isolation between redundant elements and independent power sources are not possible in todays FPGAs).

Beside a combination of two fail-silent units (which typically include two nodes each) the TMR approach allows to built a fail-operational system. These are the most expensive approaches, but they allow the mitigation of all single hardware faults (voter in TMR system might be an exception). In a TMR system reliability will decrease, if a defect affects the functionality of one module. Accordingly, a defect module has to be repaired as soon as possible. In [14] they state some advantages of using three discrete FPGAs in these TMR architecture. A fault in one of the FPGAs can be detected. In case of a transient failure, it can be corrected by reconfiguring the defect FPGA. In case of a permanent physical defect in a minor part of the FPGA, reconfiguration has to be performed avoiding the faulty area. A certain area overhead is necessary in order to provide alternative circuit blocks and to include the logic managing the reconfiguration. These techniques are not possible with MCUs respectively ASICs in general.

For an MCU based system, both the hardware and the software have to be verified. If an operating system is used, then the operating system itself must be verified for use within a safety-critical application. With an FPGA the implemented hardware has to be verified only, once the HDL has been functionally simulated and equivalence has been checked.

Many modern MCU offer multiple on chip peripherals such as memory and communication controllers and not all of these will be used in many applications. These unused peripherals must still be handled in accordance with the MCU data sheet to ensure that the unused peripherals cannot interrupt or otherwise influence the program execution. The components used to terminate these unused peripherals will contribute to the calculation of the failure rate of the design. However, this is not the case in FPGAs.

IV. Conclusion

In this overview, the impact of the decision for a specific embedded hardware platform (MCU vs. FPGA) on the system's safety was compared by investigating fault handling techniques present in these devices. The fault handling measures, namely fault avoidance and fault tolerance, are influenced by several hardware platform specific issues which were presented in Fig. 1 for hardware and software faults.

In case of transient hardware faults, MCUs and FPGAs are affected differently, but in both cases hardware faults could fundamentally change the behavior of the device. Fault mitigation in program/configuration and data memories can be applied on both platforms by hardware or software techniques. Mitigation of faults in CPU registers needs further measures, either in pure software (resulting in an immense runtime and memory overhead) or specific hardware approaches on general purpose hardware (super scalar, dual core) or on hardware specialized for safety-critical applications. Transient faults in FPGAs can be mitigated by full or partial duplication of functionalities resulting in an area overhead. Furthermore, mitigation of permanent faults is possible in FPGAs if only minor parts of the FPGA are affected and partial configuration is supported. Systematic hardware faults could be mitigated

by appropriate design processes and hardware verification capabilities on MCUs as well as on FPGAs.

Further on, differences are present between MCUs and FPGAs in the handling of software faults. These differences include the software design process, especially the languages used with their specific properties and software verification capabilities. In both cases it has to be assured that the later system is behaving as intended. In case of MCUs aspects as side effects (interrupts, memory, etc.) complicate this verification process while in the FPGA design the handling of interconnected parallel processes might be challenging. Architecture measures can be applied in MCUs as well as in FPGAs to tolerate software failures during run time. However, in FPGA design it might be beneficial that these measures usually do not affect the timing behavior of the original function.

We argue that designers of safety-critical systems should consider **all** impacts of different hardware platforms on the overall safety and reliability on their system. The survey of impacts presented in this paper for MCUs and FPGAs could be used as basis for this consideration and hopefully inspires further work in this field of safety-critical embedded systems.

REFERENCES

[1] ALTERA. Error detection & recovery using crc in altera fpga devices. In *Application Note 357*, April 2006.

[2] T. Anderson and P. A. Lee. Fault tolerance terminology proposals. In *Proceedings of the 12th IEEE International Symposium on Fault Tolerant Computing*, 1982.

[3] E. Bahl, T. Lindenkreuz, and R. Stephan. The fail-stop controller ae11. In *Proceedings of the International Test Conference*. IEEE, 1997.

[4] R. Baumann. The impact of technology scaling on soft error rate performance and limits to the efficacy of error correction. In *Digest of the Internation Electron Devices Meeting IEDM'02*. IEEE, 2002.

[5] M. Berg. Vhdl synthesis for high-reliability systems. In *2004 MAPLD International Conference*, 2004.

[6] P. Bernardi, L. Bolzani, M. Rebaudengo, M. S. Reorda, F. Vargas, and M. Violante. On-line detection of control-flow errors in socs by means of an infrastructure ip core. In *International Conference on Dependable Systems and Networks (DSN'05)*, 2005.

[7] L. Bolzani, M. Rebaudengo, M. S. Reorda, F. Vargas, and M. Violante. Hybrid soft error detection by means of infrastructure ip cores. In *10th IEEE International On-Line Testing Symposium (IOLTS04)*, 2004.

[8] D. Borrione and P. Georgelin. Formal verification of vhdl using vhdl-like acl2 models. In *Forum on Design Languages (FDL)*, 1999.

[9] S. Brewerton. Dual core processor solutions for iec61508 sil3 vehicle safety systems. In *Embedded World Conference*, 2007.

[10] J. Cotner. Mini aercam project, 2004.

[11] S. Dellacherie, L. Burgaud, and P. di Crescenzo. Improve-hdl-a do-254 formal property checker used for design and verification of avionics protocol controllers. In *Digital Avionics Systems Conference (DASC'03)*. IEEE, 2003.

[12] R. Dobias and H. Kubatova. Fpga based design of railway's interlocking equipments. In *EUROMICRO Symposium on Digital System Design*. IEEE, 2004.

[13] M. Frate, Garg and Pasquini. On the correlation between code coverage and software reliability. 1995.

[14] M. G. Gericota and J. M. Ferreira. Restoring reliability in fault tolerant reconfigurable systems. In *Actas das Jornadas sobre Sistemas Reconfigurveis (REC'2005)*, 2005.

[15] J. Hammarberg and S. Nadjm-Tehrani. Formal verification of fault tolerance in safety-critical configurable modules. *International Journal of Software Tools for Technology Transfer*, 7, 2004.

[16] A. Hilton and J. Hall. On applying software development best practice to fpgas in safety-critical systems. In *10th International Conference on Field Programmable Logic and Applications*. Springer-Verlag, 2000.

[17] A. Hilton and J. G. Hall. Developing critical systems with pld components. In *Formal Methods for Industrial Critical Systems (FMICS'05)*, 2005.

[18] IEC61508. *Functional safety for electrical / electronic / programmable electronic safety-related systems*. International Electrotechnical Comission, 1998.

[19] J. Jou and C. Liu. Coverage analysis techniques for hdl design validation. In *6th Asia Pacific Conference on Chip Design Languages*, 1999.

[20] R. Katz, K. LaBel, J. Wang, B. Cronquist, R. Koga, S. Penzin, and G. Swift. Radiation effects on current field programmable technologies. *IEEE Transactions on Nuclear Science*, 44, 1997.

[21] J. C. Knight. Safety critical systems: Challenges and directions. In *24th International Conference on Software Engineering (ICSE'02)*, 2002.

[22] J. Lach, W. H. Mangione-Smith, and M. Potkonjak. Low overhead fault-tolerant fpga systems. *IEEE Transactions on VLSI Systems*, 6, 1998.

[23] J. H. Lala and R. E. Harper. Architectural principles for safety-critical real-time applications. *Proceedings of the IEEE*, 82, 1994.

[24] N. G. Leveson. *Safeware - System Safety and Computers*. Addison-Wesley, 1995.

[25] J. McCollum, R. Lambertson, J. Ranweera, J. Moriarta, J.-J. Wang, F. Hawley, and A. Kundu. Reliability of antifuse-based field programmable gate arrays for military and aerospace applications. In *4th Military and Aerospace Applications of Programmable Devices and Technologies International Conference*, 2001.

[26] MISRA-C:2004. *Guidelines for the use of the C language in critical systems*. Motor Industry Software Reliability Association, 2004.

[27] NASA. Trst* and the ieee jtag 1149.1 interface. na-gsfc-2004-04. 2004.

[28] B. Nicolescu, R. Velazco, M. Sonza-Reorda, M. Rebaudengo, and M. Violante. A software fault tolerance method for safety-critical systems: Effectiveness and drawbacks. In *15th Symposium on Integrated Circuits and Systems Design*, 2002.

[29] F. Nicoli and L. Pierre. Formal verification of behavioral vhdl specifications: a case study. In *Conference on European Design Automation*, 1994.

[30] E. Normand. Single event upset at ground level. *IEEE Transactions on Nuclear Science*, 43, 1996.

[31] N. Oh, P. P. Shirvani, and E. J. McCluskey. Error detection by duplicated instructions in super-scalarprocessors. *IEEE Transactions on Reliability*, 51:63–75, 2002.

[32] J. Penton and S. Jalloq. Cortex-r4, a mid range processor for deeply-embedded applications. may 2006.

[33] B. Pratt, M. Caffrey, P. Graham, K. Morgan, and M. Wirthlin. Improving fpga design robustness with partial tmr. In *8th annual Military and Aerospace Programmable Logic Devices (MAPLD) International Conference*, 2005.

[34] R. B. R. Katz and K. Erickson. Logic design pathology and space flight electronics.

[35] J. J. Rodrguez-Andina, J. Alvarez, and E. Mandado. Design of safety systems using field programmable gate arrays. In *Workshop on Field-Programmable Logic and Applications (FPL'94)*, 1994.

[36] J. Rose, A. E. Gamal, and A. Sangiovanni-Vincentelli. Architecture of field-programmable gate arrays. In *Proceedings of the IEEE*, volume 81, 1993.

[37] F. Salewski and S. Kowalewski. Exploring the differences of fpgas and microcontrollers for their use in safety-critical embedded applications. In *Symposium on Industrial Embedded Systems (IES'06)*. IEEE, 2006.

[38] B. Schlich and S. Kowalewski. [mc]square: A model checker for microcontroller code. In *Int'l Symp. on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 2006)*. IEEE, 2006.

[39] N. Storey. *Safety-Critical Computer Systems*. Prentice Hall, 1996.

[40] F. Vahid and T. Givargis. *Embedded System Design - A unified Hardware/Software Introduction*. Wiley, 2002.

[41] F. Vargas and A. Amory. Transient-fault tolerant vhdl descriptions: A case-study for area overhead analysis. In *Ninth Asian Test Symposium (ATS'00)*, 2000.

[42] R. Velazco and S. Rezgui. Transient bitflip injection in microprocessor embedded applications. In *6th IEEE International On-Line Testing Workshop (IOLTW)*, 2000.

[43] M. M. Wang. Seu mitigation techniques for xilinx virtex-ii pro fpga. 2004.

[44] B. Wells and S. M. Loo. On the use of distributed reconfigurable hardware in launch control avionics. In *Digital Avionics Systems Conference (DASC)*. IEEE, 2001.

[45] M. Wirthlin, E. Johnson, and N. Rollins. The reliability of fpga circuit designs in the presence of radiation induced configuration upsets. In *IEEE Symposium on field-Programmable Custom computing Machines (FCCM)*, 2003.
[46] Xilinx. Xapp564 ppc405 lockstep system on ml310. Jan 2007.
[47] Xilinx. Correcting single-event upsets through virtex partial configuration. June 2000.