V&V TECHNIQUES FOR FPGA-BASED I&C SYSTEMS – HOW DO THEY COMPARE WITH TECHIQUES FOR MICROPROCESSORS?

Sam George and Sofia Guerra
Adelard LLP
24 Waterside, 44-48 Wharf Road,
London N1 7UX. United Kingdom.
srjg@adelard.com; aslg@adelard.com

ABSTRACT

We compare verification and validation (V&V) techniques for FPGA and microprocessor-based instrumentation and control (I&C) systems from the point of view of standards compliance, an approach based on behavioural properties, and the analysis of vulnerabilities. We found that the non-technology-specific elements of the standards considered are very similar. Differences are more marked when considering behavioural properties and vulnerabilities: the amount of effort required and confidence level obtained depend on a number of properties of the particular design under verification.

Key Words: Microprocessor, FPGA, verification, vulnerability, standard

1 INTRODUCTION

This paper considers the verification and validation (V&V) techniques that can be applied to microprocessors and FPGAs, highlighting techniques that are similarly applicable to both, and those that are specific to a particular choice of platform. In each case, the relative effectiveness and the contribution of these techniques to confidence that an instrumentation and control (I&C) system can correctly perform a Cat A function is considered. Section 2 introduces the context for the work and the approach taken. Section 3 considers similarities and differences between the V&V requirements of standards for each of the architectures. Section 4 similarly examines V&V techniques needed to establish particular facets of I&C behaviour, while Section 5 does the same for design and implementation vulnerabilities. Section 6 concludes.

2 BACKGROUND

Field Programmable Gate Arrays (FPGAs) have been gaining interest from the nuclear industry for a number of years. Their perceived simplicity compared to microprocessor-based platforms is expected to simplify the licensing approach (for some types of applications), and therefore reduce licensing risks compared to software-based solutions.

Although the use of an FPGA can result in a final product that is hardware only, with no run-time software, the process used to develop the application is software-intensive, using advanced software tools to design, implement and verify the application. In both cases there is a process of specification, coding and compilation (even if different languages and tools are used). We would therefore expect the approaches taken for justifying software-based systems to be broadly similar to the justification of FPGA applications (when these are based on the development lifecycle). Indeed, there is a growing international consensus that the regulatory review of FPGA-based systems should treat the application development process in a manner similar to software development, invoking many of the same standards and guidelines that are used for software-based systems, with some adaptation.

There have been a number of applications of FPGAs in the nuclear industry, such as the Main Steam and Feedwater Isolation System at Wolf Creek plant, in the US (class 1E), and a number of safety applications including Reactor Trip Systems (RTS) for four Nuclear Power Plants (NPPs) in Ukraine (24 systems), Engineering Safety Features Actuation Systems (ESFAS) for five NPPs in Ukraine and Bulgaria (18 sets) and Reactor Power Control and Limitation System (RPCLS) for four NPPs (8 systems). A number of applications are planned for new NPPs. These applications were developed prior to the publication of IEC 62566 [1]. For most of the applications in the nuclear industry, there was no specific FPGA guidance or standard for the development and justification of FPGA-based systems in nuclear applications. The approach taken was to adapt software regulations and standards to the context of FPGAs.

During 2014, we worked on a project funded by Energiforsk with the objective of developing an overview of the position of safety-related systems built on FPGA technology for nuclear applications. This investigated whether FPGA-based systems were a realistic alternative in future investment programs in the Nordic NPPs within the next five years, considering technological advancement, licensing, market situation etc. The conclusion was that FPGAs may have a role in future modernisation of I&C systems in Sweden [2].

The project on which this paper draws [3] was a continuation of the work performed in 2014. Its objective was the evaluation of the V&V activities that are necessary to implement an FPGA-based application and compare them with equivalent activities to assess a microprocessor-based solution.

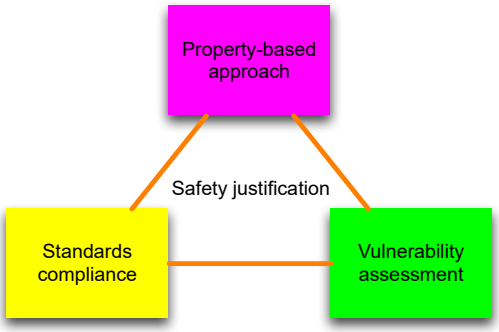

Figure 1: Strategy triangle of justification

This study reviews the V&V activities that are needed to deploy an FPGA-based product in a nuclear application and compares them with what might be equivalent for a microprocessor-based implementation. Different activities have different objectives in terms of assurance, and will achieve different levels of confidence in the system. In order to be able to perform this comparison, it is necessary to define criteria. We base our overall approach on a comparison of different aspects of a safety demonstration so that similar levels of assurance can be achieved across the different architectures[1]. This is done by considering the V&V activities required by relevant standards, verification and validation activities to achieve confidence that all behavioural properties have been realised, and verification activities to ensure that typical vulnerabilities of the technologies have been avoided. This approach follows the three aspects of assurance that we usually describe as *the strategy triangle of justification* [4].

The focus of this project is on safety functions, or those designated as category A according to IEC 61226 [5] and on "small and medium size applications" where a logic integrated circuit (IC) and its program is the main focus of assurance. It does not cover large applications where several FPGA-based

[1] We acknowledge that there is a difference in scope of the areas covered by the V&V activities discussed. In particular, some of the aspects covered in FPGA development have no equivalent in microprocessors because some of the nearest equivalent processes (for example timing analysis taking into account signal propagation) are carried out in microprocessor hardware V&V.

units communicate with each other via networks or communication links (although several of the issues would be similar).

3 STANDARDS COMPLIANCE

This section summarises our findings from our review of the V&V activities required by comparable standards for FPGA-based and software-based I&C systems performing Cat A functions. The standards chosen were IEC standards: IEC 60880 [6] for microprocessor-based systems and IEC 62556 [1] for FPGA-based systems. To identify the differences for discussion, we first carried out a side-by-side comparison of the two standards. As the prescribed development methodologies share many features, we began by looking at each distinct lifecycle phase. Where the techniques were qualitatively similar or where there was a similar process element, we went on to compare the treatments in the two standards.

Overall, there are very few significant differences in V&V requirements imposed by IEC 60880 and IEC 62566. The two major differences that we drew out were that IEC 62566 is generally less prescriptive about how the results of the different activities have to be recorded, and that some of the ambiguities in phrasing in IEC 60880 have been clarified in the more recent IEC 62566. Where IEC 60880 was previously unclear about precisely which sections were applicable in different situations, IEC 62566 has gone some way to address this.

In the requirements phase, there is little difference between the standards, except for verification of the requirements specification. Section 6.6 of IEC 62566 requires that a critical analysis of the requirements specification is performed, while IEC 60880 does not. This critical analysis is intended as a verification of the requirements specification, and provides an opportunity to identify potential omissions and inconsistencies before design and implementation begins.

In the design and implementation phase, IEC 62566 is more explicit that additional design considerations may apply on a case by case basis, with Section 8.3.4 stating that the design rules should reflect the latest knowledge. That is, IEC 60880 may be interpreted as containing an exhaustive list of all design considerations, while IEC 62566 makes it clearer that the design considerations identified are a representative sample of those that may apply for any given system.

There are no significant differences in provisions regarding software tools for development. This is one of a number of areas in which IEC 62566 incorporates stipulations from IEC 60880 by direct reference. IEC 62566 places much more emphasis on automation of tests, requiring tests to be fully automated and any manual input or observation justified (as these are considered potentially error-prone). IEC 60880, by contrast, permits automated code analysis but does not require manual analysis to be justified. This is not surprising, given that setting up and recording tests expressed in low level logic is less surveyable and more susceptible to error than analysis of ordinary software.

The requirements concerning software and hardware description language (HDL) aspects of system integration are very similar between the two standards, with IEC 62566 making explicit requirements of verification tools (Section 15). The differences are also slight in the treatment of system validation.

The acceptance criteria for pre-developed products differ markedly between IEC 62566 and IEC 60880. In particular, Section 15.3.2 of IEC 60880 explicitly identifies the documentation that we should expect to be made available. This includes the software quality plan, specification documents, software/hardware integration plan, validation plan and results of verification and validation. By contrast, IEC 62566 requires only that a documentation review be carried out of the design and verification documents of the pre-developed product. This difference could be partly attributed to the practical difficulties in obtaining information from intellectual property (IP) core originators. While IEC 60880 requires that pre-developed products themselves comply with IEC 60880, no similar provision is made in IEC 62566. Section 9.3 of IEC 62566 requires that they be developed in accordance with the rules of the suppliers that made the products, which is not an absolute verification standard. Section 9.1 of IEC 62566

identifies a requirement to confirm the adequacy of selection of pre-developed items, and of the conformance of such items with their component requirement specification – that is, the use of pre-developed items must be justified, and shown to be necessary within the wider system. The stipulations around operating experience could be construed to be tighter in IEC 62566: although Section 7.4.3 of IEC 60880 requires that such experience be in "similar" conditions, IEC 62566, in Section 7.4.3, requires that it be "equivalent".

In summary, we concluded there are no significant differences in comparable V&V requirements between IEC 60880 and IEC 62566 in their V&V requirements. It does not follow that a system developed using one technology in compliance with the relevant standard is more likely to be a correct implementation than the other. To reach a more detailed understanding of these differences requires consideration of more architecture-specific vulnerabilities and the ways functions are implemented, especially in the FPGA case. A qualitative analysis of these issues follows in the next sections.

4 BEHAVIOURAL PROPERTIES

In the context of the strategy triangle of justification, consideration of behavioural properties aims to show that the expected behaviour of the system or component is realised. Typically, such analysis is organised under attribute headings such as functionality, timing and accuracy. This usually requires the use of a number of V&V techniques. Different properties (or attributes) can be considered for different types of systems or components. The exact set of attributes to be considered should be defined for each system. Accordingly, in assessing the usefulness of a particular technique, it is necessary to consider the contribution of the behavioural attribute to the overall case, the contribution of the V&V technique to establishing the behavioural attribute, and the inputs required to apply the V&V technique.

The combination of all the techniques deployed for each behavioural attribute generates a level of confidence that the behaviour of the complete system is well understood and correctly implemented. The level of confidence achieved depends on the design under examination, the technique, and the manner and extent of its application. In this section, for common groupings of behavioural attributes, we compare how V&V techniques vary between microprocessor and FPGA-based systems, with particular emphasis on those areas where the V&V required for one architecture gives significantly more confidence or requires significantly less effort than another.

4.1 Functionality

The V&V of a system's functionality refers to the correct implementation of the defined system functions. One expression of functionality involves specifying the existence of a facility or capability of the system: for example, an instrument measuring temperature may specify that the user be able to calibrate a sensor using some particular kind of data set, perhaps over a particular interface. A part of the specification of the functionality of the system can also involve prescribing an algorithm or properties of an algorithm used to calculate some quantity in the system.

Code inspection is a useful technique to establish presence of a particular function for both conventional imperative code and HDL. However, the contribution of the technique is likely to be different in each case. Effective HDL inspection is not possible if the code involves convoluted binary logic and large module sizes. In its spatial extension of its execution space topology, it is also inherently concurrent, which can lead to overlooked timing subtleties and concurrency bugs. In addition, application functionalities are easier to understand in compact high-level languages, while HDLs are, by comparison, very low-level languages.

Random testing and functional testing are similarly applicable to microprocessors and FPGAs, where the architecture of the implementation in the black box is irrelevant.

Functional verification using formal deductive techniques or rigorous simulation is expensive, regardless of the architecture used. FPGAs, however tend to require more extensive toolsets. This is a function of the greater complexity of the domain, which must be partly modelled at an analogue level. The focus of formal proofs of functionality can be different for microprocessors and FPGAs. Specification predicates for ordinary programming languages are likely to focus on higher level features than those for synthesisable HDL, which deals more with hardware and analogue details that do not need to be considered for microprocessor software. The main risk here is that the specification against which an FPGA functional verification exercise is conducted may not sufficiently capture the specification at the application level (*i.e.*, it may be less complete). Similarly, application level model checking operates at a higher level of abstraction than ordinary programming languages or HDL, but the semantic gap is larger in the case of HDL, so it is particularly important to be confident that the model is actually realised by the underlying low-level code.

4.2 Timing

Timing behaviour has different facets depending on whether we are considering low-level hardware concerns such as propagation of signals and rise times of particular transistor technologies, length of time for a particular clocked processor to work through a compiled algorithm written in a high level programming language, or latency of a real time signal processing pipeline or response time to an asynchronously presented demand. Some of these concerns apply predominantly to microprocessor-based implementations, while some apply to HDL implementations. Functional level testing techniques apply to both.

Time response tests, which measure timing behaviour at an application level, are essentially a kind of black box testing. Worst case execution time (WCET) analysis, on the other hand, is an idea that is ordinarily only applicable to microprocessors, and relates to the elapsed time taken for control to pass from one point to another in an imperative program. Depending on the way an FPGA is programmed, spatial control flow analogues can give rise to a similar concept, but it would be unlikely to be called WCET. Static timing analysis is a technique that applies to FPGAs, insofar as there are analogue considerations in arguing their correctness. In the microprocessor case this is always a hardware issue. FPGAs have a strong advantage over microprocessors where an algorithm is computationally intensive but naturally divisible into parallel tasks and potentially produce a faster response. In these cases, raw capability of the architectures is a more significant factor than ease of V&V.

4.3 Accuracy

The definition of accuracy changes radically depending on the system boundary. For example, it can be of the digital output (of temperature, for the sake of argument), given the actual physical state of a sensor, the digital output, given the analogue potential difference across a thermocouple (this has the same limitation), the accuracy of the internal digital representation of voltage, given the potential difference across a thermocouple, the numerical precision of that voltage, the numerical precision of the output (temperature) or the numerical precision of the intermediate values in any calculations or the divergence of the result from idealised real number intermediate values. We may therefore be talking about the precision of an entire instrument, a component of it, or an algorithm. The accuracy and precision of the physical aspects of the instrument and analogue-digital/digital-analogue converter resolutions are a relatively high level issue that should be dealt with in framing the requirements of the entire software application, but the precision and numerical stability questions tend to arise at the implementation stage.

Again, black box system testing is identical regardless of the implementation technology. Accuracy test by simulation would differ in scope in microprocessors and FPGAs. Simulation for FPGAs animates lower level semantics than are involved in executing tests in high level programming languages. In the FPGA case, the main concern in providing equivalent coverage of accuracy is that adequate testing is achieved at a higher level of abstraction than gates and wires. Numerical analysis concerns discretisation

and algorithm choices: it operates at a higher level of abstraction than ordinary programming languages or HDL, and if it is used, the implementation technology makes little impact on its scope or complexity.

4.4 Availability

Availability of a system is its readiness for correct service. It is a system-level attribute supported by component attributes. In the cases of both microprocessors and FPGAs it can be considered as a hardware attribute, in which case it reduces to the physical reliability of electronic components, or a software process, for which it relates to the ability of a software system to service external interactions, which requires it to be operating properly and keeping the correct internal state. In the latter case, the speed of the hardware and freedom of the code from starvation or dead state issues are relevant, but are more appropriately considered as vulnerabilities.

4.5 Robustness

Robust behaviours are tolerant to out-of-normal inputs and stressful conditions. Where this concerns over-voltage or other physical problems, these are largely hardware properties that apply similarly to microprocessors and FPGAs, as the integrated circuit (IC) fabrication technology determines the electrical sensitivity to over- and under-voltage or inaccurate clocking. In software, it is possible that misconfiguration of scaling parameters can lead to unforeseen overflows or nonsensical values, which can then cascade meaningless computations into other processing functions. Negative testing at extremes of value ranges can provide some confidence that this should not happen. Microprocessor code and HDL should both be checked for implementation of sensible defensive behaviours, such as range checking of values before they are used. Both microprocessor based designs (where interrupts might be activated by an external stimulus more frequently than the coder expected) or FPGA designs (where some asynchronous stimulus is applied more often than expected) could lead to demands being ignored or timing out. Robust software or HDL design should have defined behaviours in overload circumstances such as this (for example, ignoring or queuing), which are validated to be safe in the context of the wider application.

4.6 Fault tolerance, diagnostics and failure recovery

Many of the system level approaches to fault tolerance apply similarly to FPGA based systems as they do to microprocessor-based systems. Familiar techniques such as the use of error detection and correction codes and modular redundancy are equally applicable. FPGAs increase the scope of some of these approaches because they can be implemented more flexibly on-chip. For example, modular redundancy on an FPGA may be implemented by spatially separating multiple instances of the same logic. This approach can work together with usual design strategies such as divisional redundancy.

Since FPGAs are dynamically configurable (with the exception of the one-time-programmable antifuse type), they are susceptible to configuration upsets as well as execution upsets. FPGAs can provide some semi-automatic detection and repair facilities. However, some of these approaches rely on IP cores and may therefore increase the design footprint of the whole solution and thus also the justification overhead.

Failure recovery differs from fault detection and tolerance attributes in that it concerns the response to a failure that has occurred rather than one that has been masked to a higher system level. Modular approaches to fault containment and recovery from failure are sufficiently abstract that there is no real difference between the approach taken with microprocessors and FPGAs. The main difference between FPGAs and microprocessors in the scope for implementing failure detection and repair of soft errors is that the level of tolerance for FPGAs can be very flexibly tailored to a given target tolerability of upsets.

4.7 Conclusions

Many V&V techniques that pertain to system level behaviours are equally applicable to microprocessors and FPGAs. However, behaviour relating to timing and concurrency needs to be assessed in different ways. FPGAs have extra behavioural facets that must be considered because verifying them involves tools that must use non-discrete physical models to handle issues such as propagation delay. These V&V obligations arise on a per-development basis rather than at chip design time as is the case for microprocessors, although in the case of microprocessors, there is in practice little prospect of gaining access to the verification records of the microprocessor itself. V&V for FPGAs does not need to address an operating system or require analysis of control flow through instruction sets at a low level, but it is important not to neglect any dataflow paradigms that might be adapted from microprocessor development idioms, which may not be naturally covered by HDL verification tools that are aimed at low level IC development.

5 VULNERABILITY ANALYSIS

Vulnerabilities of artefacts, tools or processes are properties of particular technologies that often lead to predicable patterns of failure. For example, a program written for a microprocessor will likely involve using and manipulating memory addresses, which is well known to be a source of programmer error. Concurrency in programs for microprocessors is vulnerable to data corruption, starvation or performance issues, while the synthesis and deployment of HDL code onto an FPGA is known to be vulnerable to timing propagation errors and unpredictable asynchronous behaviours. At the physical level, SRAM types of FPGAs in particular are vulnerable to spontaneous configuration changes owing to ionising radiation.

5.1 Equivalence between design and implementation levels

Development of both HDL and code for microprocessors usually proceeds according to the ordinary V model of development. Using the V model, a design is implemented in increasing detail at progressively lower levels of abstraction. With each artefact and translation between equivalent artefacts at different design levels, there are opportunities for errors to occur. V&V processes can be applied to each artefact as it is generated in order to address this vulnerability, and to the tools and processes that are used to move from a high level artefact to a more detailed one. As part of the development process, the functionality is broken down into smaller components, and these are verified at each stage within the semantics of the abstraction level in question. When all of the low level modules have been implemented, they are again verified as they are integrated towards the top system level, with validation of the final implemented system taking place against the original requirements.

In FPGA contexts, it is common to use “Electronic System Level” (ESL) tools to capture a design at a high level. “ESL” is a term used by IEC 62566 to mean “high-level description of an electronic system, based on a set of processes representing functionalities of components such as microprocessors, memories, specialized computing units, or communication channels” [1]. This is often referred to as “Transaction Level Modelling” (TLM), and is facilitated by languages such as SystemC and SystemVerilog. IEC 62566 provisions such as section 6.6.3 (in relation to ESL usage) mandate that “[t]he requirement specification shall be reviewed to check its completeness and its consistency”; this language mirrors that in IEC 60880 about requirements specification, although IEC 60880 does not provide for any ESL-like languages other than in an oblique reference to high level tools in section 14.1.1.

Section 6.5.3 of IEC 62566 states that “[t]he semantics of the languages used to express the requirement specification at ESL level may differ from the semantics of the HDL languages used during design” [1]. The reason this can present a problem is not that the semantics differ in themselves, but the equivalence relations between the semantics can be incomplete and contain ambiguities that may be resolved differently by different human implementers or transformation tools. While ESL level tools are expressive, facilities for verification beyond simulation and assertion checking are limited. IEC 62566 (in sections 6.5 and 8.5) discusses the requirements for using ESL tools, imposing similar qualification

standards on them as for the lower level FPGA toolsets. Assertion languages can be used at ESL level, although these do not give assurance of the level of coverage or completeness. IEC 62566 (at section 8.5) raises a number of provisos if HDL is to be produced by an automated toolchain.

High level code (microprocessors) or HDL (FPGAs) to physical implementation equivalence concerns the correctness of compilers and assemblers (for microprocessors) and logic synthesis and place-and-route tools (for FPGAs). In the microprocessor case, object code analysis after compilation can provide a high level of assurance that the compiler has not introduced bugs, but this is very expensive when applied to an entire code base. Owing to the closed source nature of place-and-route tools and the mechanisms for generating bitstreams and loading them onto FPGAs, it is not possible to conduct a fully equivalent exercise of this kind using an independent tool or manual inspection. However, the refinement of HDL code to synthesisable register transfer level (RTL) subsets, and synthesis process from RTL to netlist, can be readily examined (this part of the process is more analogous to high level code to assembly code compilation for microprocessors). Moreover, the standard closed source tools for FPGA synthesis will test the bitstream against RTL as a matter of routine. Whether this bitstream checking (which corresponds to a stage absent in microprocessor based designs) provides extra confidence compared to a microprocessor, or whether it simply compensates for the extra complexity present in an FPGA is a somewhat subjective question. For FPGA designs, this problem might be mitigated by synthesising HDL code onto diverse platforms and using voting arrangements on different divisions to mitigate the consequences of any FPGA tool flaws.

5.2 Timing vulnerabilities

Timing vulnerabilities in code for microprocessors mainly concern the maximum and minimum execution time of functions, which has complex dependencies on control and data flow, interrupts, compilers and microprocessor architecture. This can lead to late responses causing delays in signal propagation or protocol violations. In the worst case, it can cause subtle concurrency bugs. Application level processing delays must also be considered for FPGA implementations, but such designs have the advantage that there is no high-level language compiler or microprocessor architecture to complicate matters, and true concurrency obviates the need for interrupts.

Full dynamic semantics of HDLs are not completely defined and are in a partly analogue domain – continuously varying quantities such as rise times and latch times, distances and propagation rates need to be known for a given layout in order to determine the behaviour of arbitrary HDL in all circumstances. The exact constraints are proprietary information of the FPGA manufacturer inherent in the physical IP on the FPGA chip, and it is unrealistic to expect them to be divulged. Timing “back annotations” are the usual mechanism to add some of this information to the original HDL without making the physical architecture explicit. By confining permissible HDL programming paradigms to synchronous RTL subsets as required by IEC 62566, these vulnerabilities are eliminated in the HDL design space, although they re-emerge during place-and-route, where the place-and-route tool and post-place-and-route static timing analyser are the points where any vulnerabilities would be critical.

For verifying simple temporal properties of RTL designs, a number of techniques are available. Integrity properties of HDL designs can be analysed: assertion languages such as PSL or SVA can be used to make logical claims of the behaviour of a given piece of HDL. Code can be verified against assertions by simulation testing of assertions, model checking and formal (static) verification of assertions. In stages after logic synthesis, independent verifiability is difficult, since bitstream generation and the analogue aspects of post-place-and-route static timing analysis are dependent on modelisations of the internals of the FPGA and are opaque to the FPGA programmer. There is little scope for analysing FPGA synthesis tools themselves, as these are proprietary and closed-source.

5.3 Initialisation

The situation with FPGAs is similar to that with microprocessors. However, the state of the gates at start-up needs to be documented. Evidence also needs to be provided that the effect of the initial state of high level data in state machines and cyclic structures has been sufficiently determined.

For a microprocessor, initialisation and reset design is dealt with at hardware design-time and is opaque to the programmer: correct operation depends on the correct installation of the processor in the hardware platform. In an FPGA design, initialisation and reset must be dealt with more explicitly: state is stored across the whole design, particularly if there are cyclic structures and pipelines. The safe initialisation of all parts of the HDL design should be covered by appropriate assertions, which can be checked using FPGA verification or simulation tools.

5.4 High level code or HDL bugs

This section concerns problems with code that arise from common patterns of mistakes, that is, of the kind that a person not expert in the application would be able to spot and correct. This applies both to high level languages (for microprocessors) or HDL (for FPGAs). For microprocessor code, integrity static analysis using tools can effectively locate many common coding and logical errors. Assertion generators can provide similar coverage of specific types of problems with HDL code. There are a number of tools available that will generate these sorts of assertions with HDL, often with a claim about the level of coverage of these vulnerabilities achieved. It should be noted that neither of these kinds of tool check that design refinement has been performed correctly and that the code realises the intention of the high-level application design. This is a matter of functional verification.

5.5 Unrevealed implicit state corruption

In an FPGA, state can be captured in cyclic data flows in a way that cannot be directly read or extracted: such structures naturally occur as HDL analogues to looping and recursion in ordinary programming languages. This is a concern in implementing fault tolerance strategies, because techniques such as voting on outputs will not necessarily disclose divergent state in intermediate structures. The difficulty does not occur in microprocessors, where assembly language diagnostic routines can be used to read memory "outside" the main computation thread in a high level programming language. Designs should therefore avoid such cyclic features without building in logic to read out the state.

5.6 Silicon design errors

At the implementation level, programs for microprocessors are immune from physical timing problems on the chip, as long as the IC design has been verified and manufactured correctly. Two issues arise with this in comparison with FPGAs. First, especially for the more specialist FPGA architectures (e.g. antifuse designs), a given chip IC is likely to be less widely used than a common microprocessor that has been in production for decades, and thus may have less compelling field experience available to provide added confidence that the hard silicon is free from design bugs. Second, FPGAs have all the same Electronic Design Automation (EDA) and fabrication vulnerabilities as microprocessors, but in addition also have software tools that allow the user to do similar kinds of design verification, but parametrically in higher-level design languages. These tools are usually proprietary to given FPGA manufacturers, and are highly complex. They therefore introduce an entirely new area of vulnerability when compared with microprocessors.

5.7 Microprocessor vulnerabilities absent in FPGAs

FPGAs are free of problems with interrupts, which is a major advantage over microprocessor-based systems. Concurrent tasks are able to run without mutual interference. Although FPGAs must deal with asynchronous demands across input/output interfaces, these demands do not interfere with other running

tasks, which makes analysis of the impact of such interactions easier. Memory management is not an issue in FPGA designs unless memory banks are used in a microprocessor-style idiom, which would defeat much of the purpose of using an FPGA. Similarly, the presence of a microprocessor IP core would vitiate many of the advantages of using an FPGA.

6 CONCLUSIONS

We have found few differences in the V&V lifecycle requirements in relevant standards for microprocessors and FPGAs. FPGAs avoid features of microprocessors that are difficult to verify, but only if microprocessor IP cores and programming paradigms are avoided. Additionally, FPGAs have design vulnerabilities of their own, largely because they require (to some extent automated) consideration of analogue circuit behaviour, such as propagation delays and rise times. In simple combinatorial logic, most of these problems can be avoided, but complex synchronous designs or reliance on analogue properties such as propagation delays to implement features can result in an application that could be more difficult to verify than if implemented on a microprocessor. Complex application logic should be verified regardless of the implementation technology, and this may be easier to do directly in high level programming languages: in FPGA technology, the HDL assertions easily framed in an HDL specification language may be too low-level (especially for synthesisable subsets), requiring behavioural HDL or ESLs to achieve an adequately concise description of the application to verify it practically. This would require more tooling and more complexity.

We conclude that from a V&V perspective neither implementation technology is better than the other in any absolute sense, and that prospective implementers should consider the applications and specification paradigms they will be using as part of their design choices.

7 ACKNOWLEDGEMENTS

This work was funded by Energiforsk within the research program ENSRIC, Energiforsk Nuclear Safety Related Instrumentation and Control systems. The work was performed with our colleague Dr Catherine Menon.

8 REFERENCES

1. "Nuclear Power Plants – Instrumentation and Control Important to Safety – Development of HDL-programmed Integrated Circuits for Systems Performing Category A Functions," IEC 62566, Edition 1.0 (2012).
2. C Menon and S Guerra, "Field Programmable Gate Arrays in safety related instrumentation and control applications". ISBN 978-91-7673-112-3, Energiforsk Report 2015:112.
3. S George, S Guerra and C Menon, "Verification and validation techniques for I&C applications in Nordic NPPs". ISBN 978-91-7673-268-7, Energiforsk Report 2016:268.
4. P Bishop, R Bloomfield and S Guerra, "The future of goal-based assurance cases," *Proceedings of Workshop on Assurance Cases*, **Supplemental Volume of the 2004 International Conference on Dependable Systems and Networks**, pp. 390-395 (2004).
5. "Nuclear power plants – Instrumentation and control important to safety – Classification of instrumentation and control functions," IEC 61226 (2010).
6. "Nuclear Power Plants – Instrumentation and Control Systems Important to Safety," IEC 60880, Edition 2 (2006).