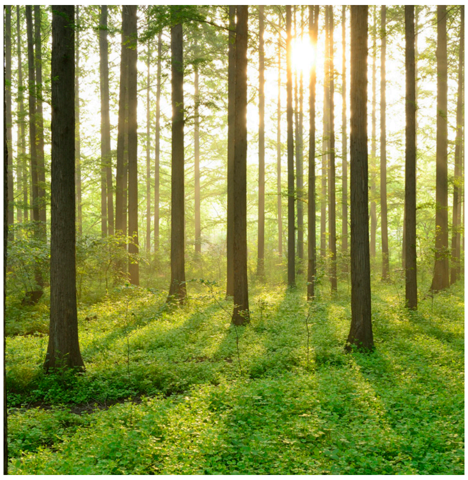
VERIFICATION AND VALIDATION TECHNIQUES FOR I&C APPLICATIONS IN NORDIC NPPS

REPORT 2016:268



ENSRIC

Energiforsk

Verification and validation techniques for I&C applications in Nordic NPPs

SAMUEL GEORGE, SOFIA GUERRA AND CATHERINE MENON, ADELARD LLP

ISBN 978-91-7673-268-7 | © 2016 ENERGIFORSK

Energiforsk AB | Phone: 08-677 25 30 | E-mail: kontakt@energiforsk.se | www.energiforsk.se

Foreword

This report is produced by Adelard LLP for Energiforsk within the research program ENSRIC, Energiforsk Nuclear Safety Related Instrumentation and Control systems. The objective of the project was to develop an understanding of the verification and validation aspects of safety related systems built on FPGA-technology (Field Programmable Gate Arrays) for nuclear applications.

FPGAs have been gaining interest from the nuclear industry for a number of years, but lately they have been questioned and the initial hypothesis that the technology would be easier to license compared to microprocessor-based platforms for nuclear applications is now questioned. This report is the second ENSRIC report on FPGA:s, and it is focused on verification and validation and standards of FPGAs compared to what is applied to microprocessor-based systems. The previous report is named "Field Programmable Gate Arrays in safety related instrumentation and control applications", Energiforsk report 2015:112.

ENSRIC is focused on safety related I&C systems, processes and methods in the nuclear industry. The three focus areas of the program are

- LTO of existing analogue platforms
- Asset management of existing digital platforms
- Emerging technologies.

The ENSRIC results are used in the plant development process, including managers, strategic teams, analysts and implementation teams at the NPPs and at the authorities, to contribute to safe and robust I&C systems that promotes low Life Cycle Cost. The program is financed by Vattenfall, Uniper, Fortum, TVO, the Swedish Radiation Safety Authority, Skellefteå Kraft and Karlstad Energi.

Sammanfattning

Den här rapporten beskriver metoder för verifiering och validering av mikroprocessorer och FPGA. Den lyfter fram metoder som är tillämpliga för dem bägge men även några som är specifika för var och en av dem.

För varje metod redogörs för dess relativa effektivitet och dess förmåga att bidra till den övergripande konfidensen på ett I&C-systems egenskaper, som utför Cat A-funktioner. Olika metoder för V&V har jämförts med hjälp av ett trefaldigt angreppssätt, som har visat sig vara effektivt vid genomförandet av en säkerhetsdemonstration. Angreppssättet beaktar på ett systematiskt sätt bidragen från kända standards, analys av beteendeegenskaper samt sårbarheter som vanligen kopplas samman med specifika teknologier och typer av design

I den komparativa analysen av olika standarder, har skillnader mellan olika relevanta IEC standarders krav på V&V, genom alla faser i livscykeln, undersökts. De signifikanta skillnader som identifierades var väldigt få. Generellt sätt kan säga att den relevanta FPGA standarden är mindre föreskrivande/normativ med avseende på vilka specifika dokument som behöver tas fram. I vissa fall görs klargöranden genom hänvisning till den motsvarande standarden för mikroprocessorer. Dessa klargöranden ställer inte krav på utförande av specifika aktiviteter utan detaljerar snarare vilka andra standarder som kan tillämpas vid specifika förhållanden. Den relevanta FPGA-standarden har också undvikit vissa av tvetydigheterna som återfinns i den relevanta standarden för mikroprocessorer.

Vid undersökning av V&V-metoderna som kan användas för att åstadkomma korrekt beteende visade det sig att många är desamma eller likartade. Särskilt snarlika är angreppssätten för dynamisk testning på systemnivå. För vissa tidsparametrar, såsom responstid och samtidighet, krävs dock olika angreppssätt. Det beror på att FPGA i grunden är en parallell modell medan mikroprocessorer är sekventiella och naturen hos de frågeställningar på den fysiska nivån som måste beaktas vid FPGA utveckling (frågeställningar som en programmerare av mikroprocessorer kan bortse ifrån). FPGA verktyg är generellt sett mycket mer sofistikerade i sitt sätt att stödja V&V jämfört med vanliga programmeringsverktyg. FPGA-verktygens ökade komplexitet är nödvändig för att kompensera för den ökade komplexiteten hos FPGA. Vid FPGA-utveckling finns ett antal frågeställningar kring elektronikdesign som behöver tas omhand, som faller inom kortdesignerns ansvarsområde i fallet med mikroprocessorer. På grund av den naturliga ensidigheten hos HDL verktyg vid hårdvaruutveckling på låg nivå kan det vara nyttigt med ytterligare kontroll av V&V aktiviteter avseende beteenden hos FPGA baserade system för att jämföra täckningsgraden för egenskaper som visar korrekt beteende på olika abstraktionsnivåer med motsvarande metoder som skulle använts vid V&V av en motsvarande mikroprocessorimplementering för samma applikation.

Metoderna för V&V som riktar sig till att analysera sårbarheten i implementeringar av FPGA respektive mikroprocesser, skiljer sig åt betydligt på grund av de radikalt olika fysiska egenskaperna och hårdvarans designparametrar och berör oftast inte systemets beteende som ”svart låda”. Många av de svårbehandlade sårbarheterna i ett mikroprocessorbaserat system så som osäkerhet kring påverkan från avbrottshanteringen på interaktionen mellan uppgifter och övergripande systemprestanda saknas vid implementering av FPGA. Många av osäkerheterna relaterade till avbrott är en konsekvens av operativsystemets växlande mellan olika

uppgifter och dess resurshantering. Operativsystems baskod är också en källa till sårbarhet eftersom den oftast är utvecklad i förväg och består till en viss del av kod vars tillförlitlighet är svår att fastställa. Omsorg måste vinnläggas om att för att lindra bristen på transparens i koden som slutligen laddas upp i FPGA. Likaledes behöver frågeställningar kring dataflöden och kontrollflöden på hög abstraktionsnivå hanteras, där HDL statisk analys inte nödvändigtvis behöver tillföra god täckningsgrad.

Kortfattat, vid jämförelse av V&V genom dessa tre olika områden, har det identifierats få systematiska skillnader bland de aktiviteter som erfordras för mikroprocessorer och FPGA baserade system. Metoderna som behövs på de lägre nivåerna är olika och kräver olika typer av expertis. För FPGA gäller i många fall att angreppssättet är mer omfattande men det är viktigt att utvärdera hela uppsättningen av de V&V –metoder som använts i varje enskilt fall för att försäkra sig om att alla nivåer i konstruktionen och implementeringen är tillräckligt täckt , speciellt om den slutliga bedömningen har gränsytor mot ett annat system som har utvecklats eller utvärderats av ingenjörers som är mer bekanta med V&V processer för mikroprocessorbaserade system. Förutvecklade komponenter, så som IP-kärnor för FPGA och operativsystem för mikroprocessorer, är viktiga aspekter att ta hänsyn till vid säkerhetsbedömning av alla I&C-system. Medan IP-kärnor kan undvikas (och de är ofta förbjudna i system som utför Cat A funktioner) kräver säkerhetsbedömning av operativsystem för mikroprocessorer en ansenlig mängd V&V-aktiviteter.

Summary

This report considers the verification and validation 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 platform architecture.

In each case, the relative effectiveness of these techniques and their contribution to the overall level of confidence of an instrumentation and control (I&C) system performing a Cat A function is considered. We have compared verification and validation (V&V) techniques using a threefold approach that we have found to be effective in achieving a safety justification. This method systematically considers the contribution of recognised standards, analysis of behavioural properties, and vulnerabilities that are commonly associated with particular technologies and design approaches.

In our comparative analysis of standards we looked for differences in V&V requirements in relevant IEC standards at all stages of the development lifecycle, and found very few significant differences. In general, the relevant FPGA standard is less prescriptive about the specific documents that need to be produced. In some cases it has clarified requirements in the equivalent microprocessor standard. These clarifications have typically not taken the form of requiring that specific activities be performed, but have given more detail on the applicability of other standards in various circumstances and have avoided some ambiguities that are present in the relevant microprocessor standard.

In examining the V&V techniques applicable to establish correct behaviours, we found that many are the same or similar. In particular, many of the system level dynamic testing approaches are identical. However, timing and concurrency require different approaches. This is a consequence of the fundamentally parallel model of an FPGA in contrast to a sequential microprocessor and the nature of the issues at the physical level that need to be considered in FPGA development (which a microprocessor programmer can neglect). FPGA tools are generally much more sophisticated in the V&V support they provide when compared to ordinary programming tools, but some of this additional complexity is necessary in order to compensate for the additional complexity of FPGAs, which must deal with a number of electronic design automation issues that are the province of the chip designer in the microprocessor case. Owing to the natural bias of HDL tools to low level hardware development, it may be a useful additional check in V&V activities for the behaviours of an FPGA based system to compare the coverage of correctness properties at different levels of abstraction with the analogous techniques that would be used in the V&V for an equivalent microprocessor implementation of the same application.

V&V techniques to address vulnerabilities in FPGA and microprocessor implementations vary considerably owing to the radically different physical properties and design parameters of the hardware, and do not generally concern the black box behaviour of a system. Many of the most intractable vulnerabilities in microprocessor based systems, such as lack of certainty about the impact of interrupts on task interaction and overall system performance, are absent in FPGA implementation flows. Many of the uncertainties relating to interrupts are a consequence of the operating system's task switching and arbitration between resources. The code base of the operating system itself is also a source of vulnerabilities, since it is usually pre-

developed and may amount to a significant amount of code whose reliability is often difficult to establish. However, care must be taken to mitigate the lack of transparency in the code artefacts that are eventually uploaded to the FPGA, as well as any data flow or control flow issues at a high level of abstraction, of which HDL assertion checking techniques do not necessarily provide good coverage.

Overall, in our comparison of V&V through these three separate lenses, we have found few systemic differences in the activities required for microprocessor and FPGA based systems, but the lower level techniques needed are different and require different types of expertise. In many cases, for FPGA techniques, the approach is more comprehensive, but it is important to review the whole suite of V&V techniques used in any particular case to ensure that all levels of design and implementation are adequately covered, particularly if the resulting justification must interface with another case that has been developed or reviewed by engineers more familiar with microprocessor based V&V processes. Pre-developed components such as IP cores for FPGAs and microprocessor operating systems are an important aspect of assuring any I&C system; while IP cores may be avoided (and are often prohibited for systems performing Cat A functions), the assurance of microprocessor operating systems will require a considerable amount of V&V activity that would need to performed.

List of content

1 Introduction

This report 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 platform choice. In each case, the relative effectiveness and the contribution of these techniques to the overall level of confidence of an instrumentation and control (I&C) system performing a Cat A function is considered. Section 2 introduces the project context and approach. 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 considers issues particular to different FPGA types. Section 7 concludes. Section 8 gives the abbreviations used and Section 9 is a list of references.

2 Background

Field Programmable Gate Arrays (FPGAs) have been gaining interest from the nuclear industry for a number of years. Their simplicity compared to microprocessor-based platforms is expected to simplify the licensing approach, 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. 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 has 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 4 Nuclear Power Plants (NPPs) in Ukraine (24 systems), Engineering Safety Features Actuation Systems (ESFAS) for 5 NPPs in Ukraine and Bulgaria (18 sets) and Reactor Power Control and Limitation System (RPCLS) for 4 NPPs (8 systems). A number of applications are planned for new builds. These applications were developed prior to the publication of IEC 62566 [6]. For most of the applications in the nuclear industry, there was no specific FPGA guidance or standard for the development and justification of FPGAs 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 with the objective of developing an overview of the position of safety-related systems built on FPGA technology for nuclear applications. This investigated if FPGA-based systems were a realistic alternative in future investment programs in the Nordic NPPs within the next 5 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.

This project is a continuation of the work performed in 2014. Its objective is 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.

This study reviews the V&V activities that are needed to implement an application in an FPGA based product and compares it with what might be equivalent for a microprocessor based application. 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. This is done considering

- the verification and validation activities required by relevant standards (Section 3)

- verification and validation activities to achieve confidence that all behavioural properties have been met (Section 4)
- verification activities to ensure that typical vulnerabilities of the technologies have been avoided (Section 5)

This approach considers the three aspects of assurance that we usually describe as *the strategy triangle of justification* [3], which is described in more detail in Appendix D.

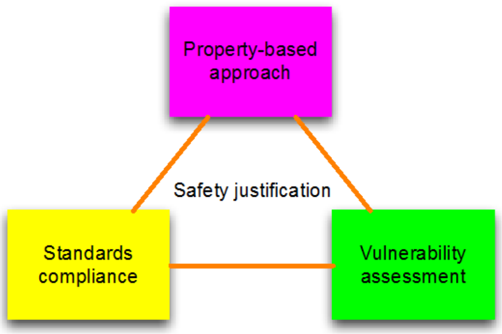


Figure 1: The strategy triangle of justification

The focus of this project is on safety functions, or those categorised as Cat A according to IEC 61226 [2], and on "small and medium size applications". It does not cover large applications where several FPGA based units communicate with each other via networks or communication links (although several of the issues would be similar). When practicable, we indicate tools that are available to support the activities.

3 Standards compliance

This section summarises our findings from reviewing 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, with IEC 60880 [4] being selected for software systems and IEC 62556 [6] for FPGA-based systems.

3.1 METHODOLOGY

To identify the differences for discussion, we first performed a comparison review of the two standards. As the prescribed development methodologies share many commonalities (Figure 3 and Figure 4), we began by looking at each distinct lifecycle phase. It is important to note that this was a semantic review and comparison as opposed to a syntactic examination; we have identified only those differences that impact the development activities, rather than cosmetic differences.

In each case where a difference was identified, we assessed this for its potential impact on the V&V activities. In some cases we considered that there would be no impact, and therefore have not taken this further in this report. Where we consider that there is a potential impact on V&V activities, we have identified this and its potential effects in Section 3.4.

3.2 IEC 60880

The scope of IEC 60880 is to provide requirements for the software aspects of computer-based I&C systems performing Cat A functions in nuclear power plants. It was first issued in 1986 and has since been re-issued a number of times to take into account the changing practices and techniques of software engineering.

IEC 60880 is directly referenced by IEC 61513 [5], which focuses on general requirements for I&C systems performing functions important to safety in NPPs. IEC 60880 is also associated with IEC 62138, which covers computer-based I&C systems performing category B and C functions.

IEC 60880 assumes a system safety lifecycle equivalent to that discussed in IEC 61513, and shown in Figure 2.

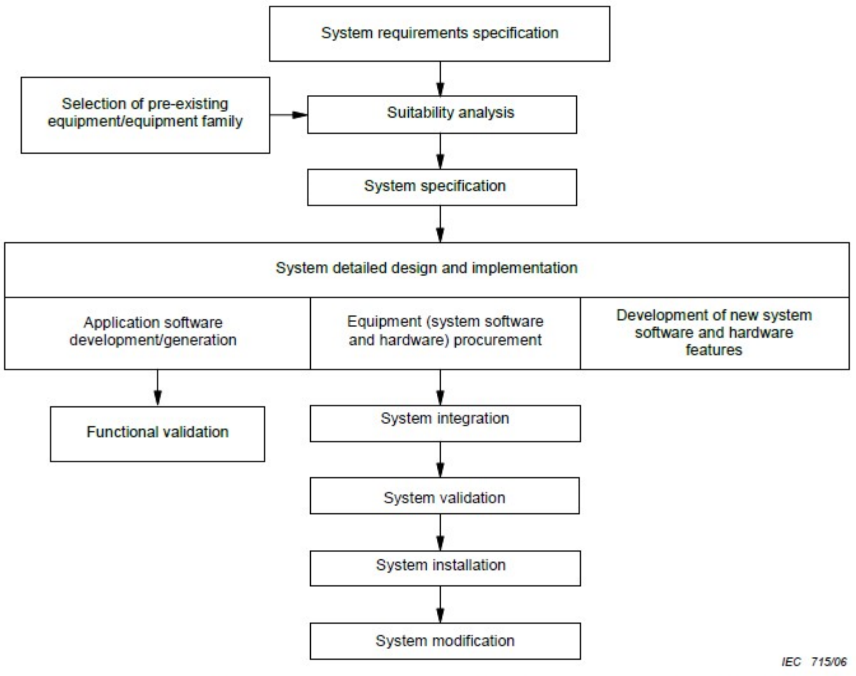

System requirements specification
Selection of pre-existing equipment/equipment family
Suitability analysis
System specification
System detailed design and implementation
Application software development/generation
Equipment (system software and hardware) procurement
Development of new system software and hardware features
Functional validation
System integration
System validation
System installation
System modification
IEC 715/06

Figure 2: System safety lifecycle from IEC 61513 [5]

IEC 60880 refines this further to identify a software development lifecycle making use of distinct phases. This is shown in Figure 3. As can be seen, each separate lifecycle phase involves verification of the phase outputs.

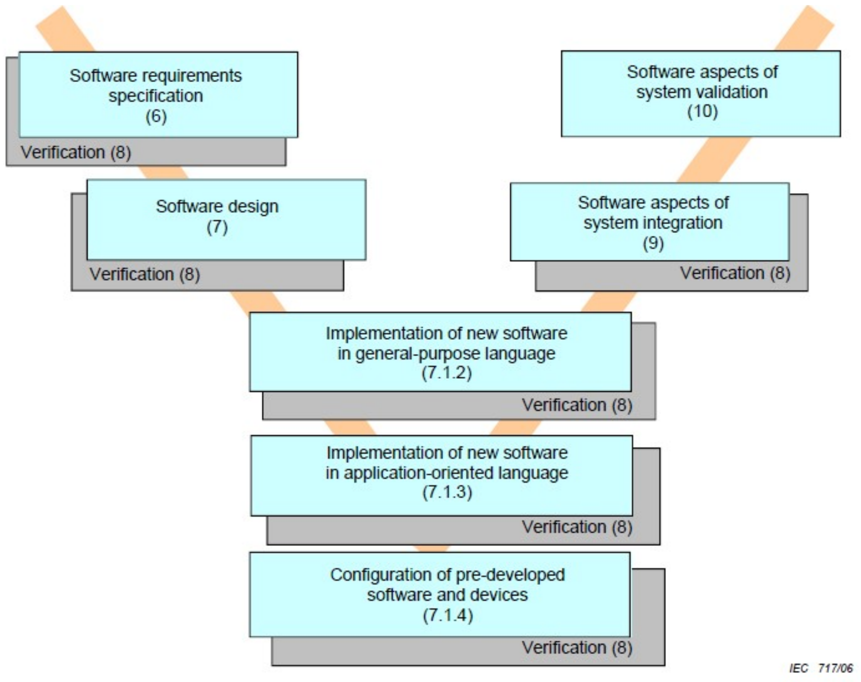

Software requirements specification (6)
Verification (8)
Software design (7)
Verification (8)
Software aspects of system validation (10)
Software aspects of system integration (9)
Verification (8)
Implementation of new software in general-purpose language (7.1.2)
Verification (8)
Implementation of new software in application-oriented language (7.1.3)
Verification (8)
Configuration of pre-developed software and devices (7.1.4)
Verification (8)
IEC 717/06

Figure 3: Software development lifecycle from IEC 60880

3.3 IEC 62566

The scope of IEC 62566 is to provide requirements for the use of HDL-programmed devices (HPDs) in I&C systems performing Cat A functions in nuclear power plants. It was first issued in 2012 and has not been re-issued since.

IEC 62566 references IEC 61513 [5], which focuses on general requirements for I&C systems performing functions important to safety in NPPs. IEC 62566 is intended to be used in conjunction with IEC 60987, which covers generic hardware design issues, and with IEC 60880 for aspects of the development when the HPD and software issues are identical.

Like IEC 60880, IEC 62566 assumes a system safety lifecycle equivalent to that discussed in IEC 61513, and shown in Figure 2. The HPD development activities as described by IEC 62566 also follow a V-model very similar to that in IEC 60880, as shown in Figure 4.

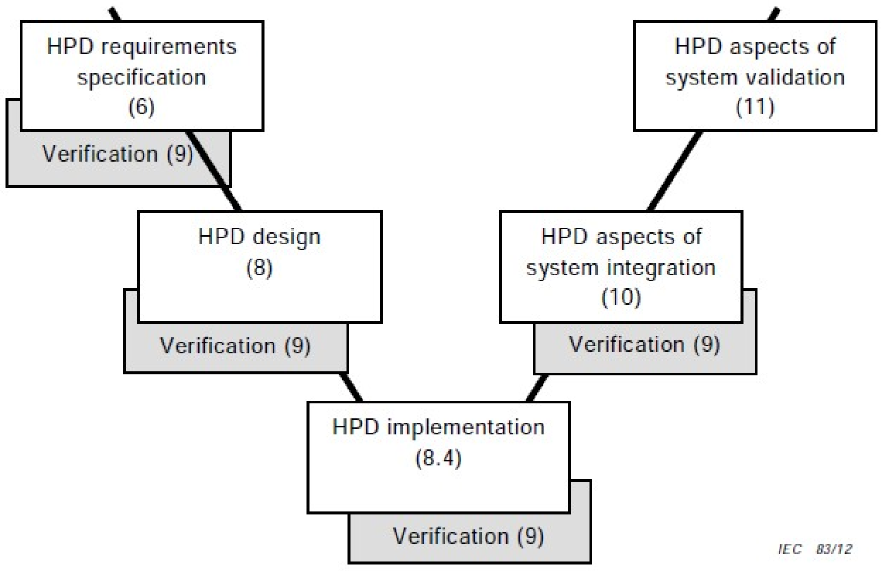


Figure 4: HPD development lifecycle from 62566

3.4 RESULTS

Overall, there are very few significant differences in terms of V&V requirements imposed by IEC 60880 and IEC 62566. The two major differences that we found were firstly, that IEC 62566 is generally more goal-based and less prescriptive about how the results of the difference activities have to be recorded and secondly, that some of the ambiguities in phrasing in IEC 60880 have been clarified in the more recent IEC 62566.

IEC 62566 appears to move away from the more prescriptive pattern of IEC 60880, and this means that alternative verification and development activities may be introduced.

Where IEC 60880 was previously unclear about precisely which sections or clauses were applicable in different situations, IEC 62566 has gone some way to address this. This is typical of what we would expect from a standard which has been developed more recently.

In the following sections we discuss our results in more detail. When identifying these, we have made the assumption that the differences of interest are those which have an impact on V&V. Where there is a difference between the standards which we consider will not have a significant impact on the V&V activities to be performed, we have not discussed this in detail.

3.4.1 Requirements phase

The requirements phase is dealt with in Section 6 in IEC 60880 and Section 6 in IEC 62566. Although the standards are slightly different, this is not due to the difference in technologies used. More specifically, there are very few significant differences relevant to this phase which impact on V&V activities, with the exception of 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 verification of the requirements specification, and provides an opportunity to identify potential omissions and inconsistencies before design and implementation begins.

We would note that Section 8.1.8 of IEC 60880 does state that the output of each phase of the development lifecycle should be verified, which could be read as a requirement to review and analyse the requirements specification. However, this is less explicit, and we consider that it would be reasonable to expect that less rigorous analyses of requirements may be presented under IEC 60880 than IEC 62566.

3.4.2 Design and implementation phase

Design and implementation is dealt with in Section 7 in IEC 60880 and Section 8 in IEC 62566. There is some divergence in the content of the standards in this section, as we might expect due to the use of design and implementation techniques relevant to the particular technology under consideration. However, not all of these differences impact V&V; those that do are described below.

In terms of design constraints, IEC 60880 is more prescriptive in what must be considered at the design and implementation stage. Specifically, both Section 7.3 and Annex B require that consideration is given to decomposition into modules, use of interrupts, execution time calculations, modification control, coding rules, memory access and so on. If this consideration is not given, then IEC 60880 requires that a justification be provided. In addition IEC 60880 also states in Section 7.2.2 that the choice of language should not prevent the use of certain error-limiting constructs, and provides guidance for the selection of language and tools. This should be contrasted with Section 8.3.4.3 of IEC 62566. This section introduces strongly recommended constraints relating to side-effects, resources, initialization of signals and delays. However, the standard emphasises that these are not mandatory, a declaration which is missing from the constraints given in Section 7.3 of IEC 60880.

In addition, 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 which may apply for any given system.

The second area in which the standards differ relates to the use of tools. IEC 60880 requires firstly (Section 7.2) that translators should be thoroughly tested, and in addition to this recommends the use of automated tools and the imposition of further requirements for tool qualification (discussed in this report in Section 3.4.7). By contrast, the design and implementation section of IEC 62566 does not explicitly require the thorough testing of tools – such as those used for synthesis, place and route – if this has already been performed and documented by the supplier of such tools, although it does also impose equivalent requirements to those discussed in this report in Section 3.4.7.

In addition to the requirements placed on tools, the standards differ in what they require to be present in the design documentation. Section 8.3.10 of IEC 62566 is more explicit than the equivalent Section 7.4 of 60880, requiring the description of design decisions pertaining to issues including control flows and data paths, protocols and algorithms, initialization of registers and the memory map. By contrast, IEC 60880 requires only that "sufficient detail" should be provided. This has two potential impacts on the verification and validation activities: firstly, that under IEC 60880 it may be possible to have multiple interpretations of how much detail is sufficient, and secondly that under IEC 62566 the adequacy of the design decisions must be justified – and can therefore be confirmed.

The final area in which the standards differ in this phase is in the extent of V&V that they recommend performing. Although both of them have a dedicated verification section (see Section 3.4.3 in this report), both also discuss V&V activities, to differing extents, in this section.

In Section 7.3, IEC 60880 explicitly requires verification of intermediate design products, which is not required by IEC 62566. We note that this may be implied in the requirement of IEC 62566 that the design should allow easy verification, but this is not equivalent to the explicit requirement that intermediate design products should undergo verification. By contrast, Section 8.7 of IEC 62566 describes a formal review process to be undertaken at the end of the design and implementation phase; this constraint is, however, omitted from IEC 60880. There is a brief mention of a review process in Section 7.4 of IEC 60880, but no equivalent description of the V&V activities to be undertaken during this. The effect of these different requirements is that we may expect different verification artefacts from the design and implementation phase of systems developed under the two standards.

Additionally, Section 8.4.7 of IEC 62566 explicitly requires static timing analysis to be performed, for which no equivalent requirement is found in IEC 60880. Correspondingly, it would be expected that verification of best and worst case time – amongst other properties – has been performed for systems developed under IEC 62566, but not necessarily for those under IEC 60880.

3.4.3 Verification

Verification of software and HPDs is dealt with in Section 8 of IEC 60880 and Section 9 of IEC 62566 respectively. Although these sections are relatively extensive, there are only a few significant differences between the two standards in this phase.

The first difference relates to the scope of verification, with particular reference to the use of pre-developed items. Section 9.3 of IEC 62566 identifies that part of the role of verification is to assess pre-developed products against the rules specified by their suppliers, and the requirements of Section 7 (dealing specifically with pre-developed items, and covered in this report in Section 3.4.8). By contrast, although Section 8.2.3.3 of IEC 60880 also requires that the requirements of Section 15 (dealing specifically with pre-developed items, and covered in this report in Section 3.4.8) are met, there is no equivalent requirement that pre-developed items should be assessed against rules specified by their suppliers. The effect of this omission is a slightly different scope in the V&V activities required by each standard. Furthermore, Section 9.1 of IEC 62566 identifies a requirement to confirm the adequacy of selection of pre-developed items, and 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. These differences should be taken into account when considering the implications of IEC 62566 Section 7 vs IEC 60880 Section 15 on verification of pre-developed items (as we do in this report in Section 3.4.8).

Secondly, we have already seen that the two standards differ in the extent to which they explicitly describe the content of documentation. IEC 60880 is more prescriptive about the documentation to be produced during verification, requiring a distinct software test specification, test report and design verification report. It is also quite specific about the information to be included in each of these, requiring for example that the test specification includes test environment, test procedures, acceptance criteria and so forth. By contrast, IEC 62566 does require that tests, goals, expected results, acceptance criteria, inputs and outputs etc. should be recorded, but does not constrain the type of documentation to be produced (e.g. test specification, test report). Although the two standards do require the same rigour of documentation in the verification phase, it is important to be aware that this documentation may be presented differently under the two standards.

In addition, 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.

Finally, with reference to the actual activities performed during V&V, IEC 62566 is slightly more prescriptive in terms of identifying the verification activities which must be performed (for example, static verification activities such as type / syntax checking, parameter checking, OOR checking and dead state detection). By contrast, IEC 60880 provides an informative annex detailing potential verification activities including program analysis, program proving, path testing, data movement testing. However, because these are informative only, their selection must be on a case-by-case basis, unlike the mandatory activities prescribed by IEC 62566.

3.4.4 Software / HPD aspects of system integration

Software and HPD aspects of system integration are dealt with in Section 9 of IEC 60880 and Section 10 of IEC 62566. There are no significant differences which impact V&V, with the only exception being that IEC 62566 explicitly requires that verification software tools should be compliant with its requirements on software tools for development (Section 15 of IEC 62566, which is detailed in this report in Section 3.4.7). IEC 60880 does not explicitly require this, so should any software tools be used for verification then this may be an area where further work could be merited.

3.4.5 Software / HPD aspects of system validation

Software and HPD aspects of system validation are dealt with in Section 10 of IEC 60880 and Section 11 of IEC 62566. As above, there are no significant differences to discuss, with the exception of a slightly more stringent constraint on the equipment used for calibration. IEC 60880 requires that this be demonstrated to be suited to the purpose of system validation, while IEC 62566 does not require this. However, this is unlikely to have a significant impact on V&V activities.

The only other area of difference in this phase relates to the documentation. Section 10.3 of 60880 identifies that software tools used in the validation process should be documented as an item in the validation report. The corresponding section dealing with validation reports in IEC 62566, Section 11.4, does not mention this, although it is included in IEC 61513. There is unlikely to be any significant impact on verification and validation, but this omission from IEC 62566 may be an indication that information on software tools for validation could potentially be missing.

3.4.6 Modification

Software and HPD modification is covered in Section 11 of IEC 60880 and Section 12 of IEC 62566. There are no significant differences between the two standards in terms of verification and validation activities.

3.4.7 Software tools for development

The use of software tools for development is addressed in Section 14 of IEC 60880 and Section 15 of IEC 62566. There are no significant differences to discuss; indeed, IEC 62566 explicitly requires conformance with IEC 60880 with the exception of a small number of constraints specific to microprocessors. It also adds some HPD-specific requirements around tools for logic synthesis, HDL source statements, command-line arguments

3.4.8 Acceptance of pre-developed products

The acceptance process for pre-developed products is considered in Section 15 of IEC 60880 (and configuration of these in Section 7.1.4), and in Section 7 of IEC 62566.

One area in which the two standards differ, and which can have a significant impact on verification and validation, is in the evaluation of the quality of the pre-developed product. IEC 62566 allows more scope for interpretation in the ways in which the quality of the product could be demonstrated. 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, the specification documents, the software / hardware integration plan, the validation plan and the results of verification and validation. By contrast, IEC 62566 requires only that a documentation review is carried out on the design and verification documents of the pre-developed product. This difference can be partly attributed to the fact that pre-developed products - such as IP cores – for use in FPGA-based systems may not be provided with all of the specific documentation discussed in IEC 60880; that is, the information may be present, but provided in a different form.

Furthermore, IEC 60880 imposes a further quality requirement, that the development of the pre-developed product should have been in accordance with the annexes of IEC

60880 itself. IEC 62566 does not impose an equivalent requirement. However, it is worth nothing that the programming process for blank integrated circuits is required to be “fault free”.

The other area in which IEC 62566 and IEC 60880 differ is in the scope for, and constraints placed on, the use of operating experience when assessing a pre-developed product. In general, IEC 62566 allows more scope for interpretation in the ways in which operating experience can be used, with Section 7.4.3 stating that it may be used to compensate for limited documentation weaknesses regarding reliability or design. By contrast, Section 15.3.3.2 of IEC 60880 is explicit that operating experience can never completely replace documentation evaluation. IEC 60880 is also more specific about the weaknesses in design that can be compensated for with operating experience.

By contrast, however, IEC 62566 is more specific about the conditions under which operating experience can be considered valid, with Section 7.4.3 requiring “equivalent” conditions. IEC 60880, in Section 15.3.3.1 requires “similar” conditions. As there is no further information given about the ways in which the conditions must be similar, IEC 60880 has the potential for multiple differing interpretations when determining the validity of operating experience.

3.5 CONCLUSIONS

Overall, we have found very few indications of significant differences impacting on V&V between IEC 60880 and IEC 62566, at any stage of the development lifecycle.

In general, IEC 62566 is less prescriptive about the specific documents that need to be produced. In some cases IEC 62566 has clarified requirements in IEC 60880, resulting in a greater specificity in these clauses. As above, these clarifications have typically not taken the form of requiring that specific activities be performed, but have rather been clarifications that, for example, the constraints of a particular clause either in IEC 62566 or in another standard are applicable in situations which were previously ambiguous.

4 Behavioural properties

In the safety triangle of justification, consideration of the behavioural properties aims to show that the expected behaviour of the system or component is met. Typically, this is organised under attribute headings such as functionality, timing and accuracy. This typically 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 would need to be defined for each system.

Therefore, 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
- 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. In selecting appropriate V&V techniques, the most productive are those that provide a high level of assurance but require modest amounts of effort, while the least attractive provide little assurance and involve large amounts of effort. The position of a technique on the spectrum depends on the nature and structure of the application being assessed. 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 type 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 sensor X 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.

V&V area	**Microprocessor V&V**	**FPGA V&V**
Code inspection	Code inspection of code written in a high level language can reveal the logical intent of well-structured single-threaded code, but by suggesting structures, may lead the reviewer into making the same logical errors as the developer. Concurrent functional requirements are difficult to analyse using code review without tool support, particularly as it may not be obvious that another function implemented elsewhere in the code may interact with shared resources such as global variables. **Effectiveness/cost** Low cost, reasonable confidence if code is well structured and commented with limited concurrent behaviour or assembly code.	Code inspection of HDL can produce less confidence than the equivalent activity for microprocessor code because hardware implementation details are exposed, and HDL designs are inherently more concurrent. It may be more useful to concentrate on code review at the ESL level rather than at executable HDL level, although subsequent coverage of potential issues that are created at HDL level only depends on the means by which the HDL is generated. A summary of the development flow can be found in Appendix B. **Effectiveness/cost** Low cost, less confidence than microprocessor, especially if review confined to low level HDL.
Random testing, functional testing	These techniques are similarly applicable to both microprocessors and FPGAs. **Effectiveness/cost** Relatively low cost. The effectiveness of testing is heavily dependent on the linkage between a test profile and the specific claims supported by the results.	

V&V area	Microprocessor V&V	FPGA V&V
Formal verification	Static analysis using a tool such as Malpas or Frama-C produces a deductive proof that a piece of code has a behaviour that is stated in a formal specification language. The effectiveness of the technique depends on the identification of significant functions and correct translation of their natural language requirements into mathematical definitions of behaviours. Depending on the extent of the properties proved, levels of confidence can be high. Tools require qualification, and those that need extensive manual intervention to link deductions together may introduce opportunities for error. **Effectiveness/cost** Depending on the functional properties verified, static analysis is time consuming but is considered to produces the highest practical levels of confidence in the functional correctness of code. Tools range in cost from free to expensive.	Assertion languages (which can sometimes be part of an HDL) enable statements to be written about the behaviour of a piece of HDL code, which can be checked by simulation or deductive means using a range of tools. However, the low level nature of these assertions can make it difficult to frame high level functional properties. Verification tools at ESL/TLM level address functionality more fully, but tend to work by simulation rather than deductive means, as the complexity of ESL language complexity does not easily facilitate deductive proof. Simulation cannot typically provide complete coverage of a state space in systems of more than trivial complexity, although there are established techniques available to gain confidence that a given level of coverage has been achieved. **Effectiveness/cost** HDL verification tools can be expensive, even on a subscription basis. These tools are generally closed source, but have long development pedigrees and wide use in industry. The level of confidence that can be obtained through using these tools at a low level for *functional* (as opposed to integrity) properties is not particularly high, but their use is mandated by IEC 62566. Some HDL verification tools provide assertion generators or check that assertions cover particular properties, but these tools cannot infer the designer's intention about what the code is supposed to do. HDL assertion checking is a technique better suited to addressing vulnerabilities in designing code for FPGAs, so we return to it in Section 5.
Model checking	Model checking techniques use a high level representation of a design in a language such as Promela and check properties of interactions between functional blocks and state transitions at a high level. In both the case of microprocessor and FPGAs, some element of skill and discretion is needed in manually constructing a suitable model. It is time-consuming and expensive task, although many common tools are free or open source. **Effectiveness/cost** In each case, the strength of the conclusions with reference to the model being checked is high, but is limited in the context of the actual implementation it is modelling by constraints on the confidence in the equivalence of the models to the implementation in question. Model checking is labour-intensive and expensive.	

Table 1: Functionality V&V

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
- 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. These issues are presented in Table 2 and elaborated in the sections that follow.

V&V area	Microprocessor V&V	FPGA V&V
Worst case execution time analysis (WCET)	We describe the issues involved in worst case execution time analysis in Section 4.2.1. **Effectiveness/cost** WCET analysis can give high levels of confidence in adequate timing performance of an algorithm implementation, but it is an expensive activity, often requiring manual intervention that is frequently intractable, in which case approximations can be used (which give a reduced level of confidence compared with an analytical solution).	Although not directly applicable as WCET analysis is a technique used for microprocessor code, HDL program code can nevertheless be written in a paradigm in which data flows round a (spatial) loop several times, and after some number of iterations emerges into a different part of a data processing pipeline. It is important to ensure that this kind of hidden but semantically significant construction is not neglected in HDL verification.

V&V area	Microprocessor V&V	FPGA V&V
Static timing analysis	For a microprocessor, verification of timing properties at the HDL physical levels takes place when the microprocessor is designed and verified by the manufacturer. The programmer writing code for a microprocessor should address WCET and performance issues, but does not need to be concerned with propagation or other electronic issues: the microprocessor programmer's model of hardware is a discrete abstraction.	Background to static timing analysis is given in Section 4.2.2. Static timing analysis is important in order to check that synchronous timing constraints that are implicit at HDL level are not violated during place-and-route. It is an automated process supported by a wide variety of tools. So called "back annotations" imposing extra constraints on the HDL as a result of how that HDL has been synthesised and placed onto the FPGA can result in a cyclic FPGA development process, where multiple revisions may be necessary before an acceptable bitstream is produced. However, this is not so different from the case with a microprocessor, where a design might not necessarily fit into memory or within the timing constraints of the application after a first pass. Similarly, with a microprocessor, some elements of the analysis of high level code or synchronous HDL can be applied independently of a particular microprocessor or FPGA, while issues such as place-and-route that are dependent on a particular FPGA are analogous in difficulties encountered when a traditional piece of code is compiled for a different microprocessor. However, in the case of a microprocessor, the extra WCET analysis that would be required for a microprocessor based design would be a larger task than rerunning an automatic static timing analysis for an FPGA. While static timing analysis for an FPGA is common practice among developers of FPGA-based solutions, WCET analysis is typically only part of rigorous development processes. **Effectiveness/cost** Static timing analysis should not pose a significant extra difficulty unless a design is being targeted onto an FPGA that is not large or fast enough to accommodate the high level design. Reliance on complex closed source toolchain may have a small negative impact on confidence when compared to a microprocessor where the consequences of place-and-route are concerned because, like a microprocessor, an FPGA *chip* design must be verified, so the verification of the HDL for the programmable part of the FPGA is an extra step with opportunities for bugs or inaccuracies. Since static timing analysis is a routine activity in FPGA development, some extra V&V benefit may be gained over microprocessors, although this is highly dependent on whether the compared code is written at a high or low level of functional abstraction.

V&V area	Microprocessor V&V	FPGA V&V
Time response test	This is a generic technique for asynchronous applications. At this level, timing properties are dictated by communications protocols or plant need. For example, real time protocols can require responses within a fixed interval, while a plant need might specify a maximum time that should elapse between a detectable set of plant conditions and a control or safety response, such as an alarm annunciation. **Effectiveness/cost** These techniques are inexpensive and give reasonable levels of confidence, although this is dependent on arguing that there is adequate coverage of the tests over inputs and environments to which timing might be sensitive. This can be more difficult in the microprocessor case, where operating system interactions between concurrent tasks often produce intractable corner cases.	

Table 2: Timing V&V

4.2.1 Worst case execution time

Most programming languages used in the development of I&C systems are *imperative languages*. This means that they set up a set of variables with state, and consist of sequential *statements* that modify that state. (The allocation of variables that form part of the notional state at any one time changes as functions are entered and exited, but this can be overlooked for the sake of the current discussion.) Some statements contain logical tests, such that the next statement to be executed may be determined by the value of some data in a variable. There is an infinite number of ways[1] of implementing an algorithm defined by an application in imperative code. Each different way of implementing the algorithm may give rise to a different number of statements that must be executed until the eventual result is computed; this number of statements will usually change, depending on the values of the input data to a given algorithm. The system of state transitions is discrete: each transition is notionally instantaneous, and any execution history always contains a whole number of executed statements. However, each of these transitions cannot in practice be instantaneous, so a given algorithm takes a finite amount of time to complete. The *worst case execution time* (WCET) is determined by whichever input data produces an execution with the largest number of statements. The more transitions, the longer this time will be. The state transitions corresponding to this timescale of sequential C statements does not have a straightforward relationship to physical time. This is because different statements will be converted into different sequences of microprocessor instructions by different compilers, and different microprocessor instructions take different numbers of *physical* clock cycles to complete. Instruction retries and processor optimisations such as pipelining, out-of-order execution and asynchronous communications with external ICs over system buses can even make the length of time needed to process an instruction non-deterministic.

In themselves, these factors make WCET analysis a complex task that involves consideration of interactions between high level language, compiler and microprocessor. It is made even harder when an operating system is used, or where function invocation is driven by asynchronous interrupts, because function execution can be arbitrarily interrupted by other parts of a system, which can result in complex mutual dependencies or reliance on unbounded external stimuli, which can result in an analytically intractable implementation. Where bounds can be found, they are often

[1] though a finite number of *good* ways

unnecessarily or unusably conservative. In such cases, it is necessary to rely on approximate or statistical techniques based on testing. Although design rules restricting the choice of microprocessor and language use can in theory make this timing behaviour more predictable, common microprocessor based platforms have all of these complexities.

The time between idealised transitions in an application level model places an upper bound on what WCET is acceptable in any given instance. Where it is decided to undertake a WCET analysis for a microprocessor based systems, tools such as AbsInt aiT can be used.

For most purposes in the analysis of a microprocessor code, the semantic model of the microprocessor is considered to be time-deterministic. However, incomplete information about any optimisations or integration into a platform which does not synchronously constrain buses or interrupts may nevertheless render the platform as a whole non-deterministic. Non-determinism may be a result of a specification being not fully known or defined, or may be a feature of a physical design that results in behaviour that is impossible to predict because a design is metastable or otherwise sensitive to probabilistic physical effects.

In summary, worst case execution time analysis is a technique used for microprocessor code: it is difficult and often intractable, so the lack of an obviously similar kind of analysis needed for FPGAs is an advantage for FPGA based designs. However, if an HDL design uses a control or data flow paradigm from a high level language for a microprocessor (such as C), some high level elements of WCET-style analysis may be appropriate, using suitable assertions, model checking or theorem proving. Even so, such an analysis should be much less affected by sources of non-determinism present in the microprocessor case.

4.2.2 FPGA static timing issues

FPGA design rules discussed below constrain HDL to a synchronous (deterministic) subset, but similarly may have non-deterministic elements at the system level, depending on how the FPGA is connected to peripheral ICs. At synchronous level, where the semantics assume instantaneous propagation, timing issues are relatively straightforward. However, there is a need to consider clock recovery tolerances at I/O interfaces and clock domain crossing between components that may be asynchronous at application level and not share a common clock. A basic V&V requirement here is to verify that the HDL design is actually synchronous. This can be checked by using an analysis tool.[2]

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 as above)

[2] to verify that the directed graph of component interconnections contains no cyclic structures that are not interrupted by a clocked register

- 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)
- 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 ADC/DAC 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. The different types of V&V activities applicable are summarised in Table 3.

V&V area	Microprocessor V&V	FPGA V&V
Accuracy tests of black box system	No differences between microprocessor and FPGA case – external measurement equally applicable in both cases. **Effectiveness/cost** Good degree of confidence at relatively low cost.	
Simulation	Applicable in both cases. **Effectiveness/cost** Highly dependent on the amount of simulation performed and the state space needed to be covered. The degree of concurrency in an FPGA can make simulation a computationally intensive task. The software needed for performing simulation testing for HDL and FPGAs tends to be more expensive than that for code in an ordinary programming language, although the facility can be part of or an extension to software that is needed in any event for other parts of the FPGA development process.	
Numerical analysis	Techniques such as sensitivity analysis and proof in interval arithmetic could be used in both cases. Where there is a temporal aspect to the accuracy of a calculation (for example, the size of a discrete interval in the numerical integration of a property over time) there is little difference between a microprocessor and FPGA, unless in one case source code is not available for a pre-developed component. **Effectiveness/cost** Sensitivity analysis can be easy and inexpensive, though relating the coverage achieved to the confidence gained can be more difficult. Proofs in interval arithmetic are difficult, time consuming and expensive, and not often done: however, they do offer very high levels of assurance.	
	The numerical accuracy of an algorithm written in a high level programming language will depend on the size of datatypes on the target architecture, the treatment of which may vary according to the compiler and targeted hardware. If a binary library is used, this information may not be readily accessible: object code can be decompiled and analysed, but this is not straightforward and may be in breach of licence terms.	FPGA data sizes and encodings are more explicit unless intermediate values or algorithmic details are hidden in pre-developed blocks (IP cores).

Table 3: Accuracy V&V

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 issues are relevant, but are more appropriately considered as vulnerabilities. On a system scale, availability can be affected by the reliability of components, which can be assessed by statistical testing. Statistical testing can be expensive where very large numbers of tests may be needed to reach adequate confidence levels.

4.5 ROBUSTNESS

Robust behaviours are tolerant to out-of-normal inputs and stressful conditions. Where this concerns over-voltage or other physical problem 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 malconfiguration of scaling parameters can lead to unforeseen overflows or nonsensical values, which can then cascade meaningless computations into other processing functions. 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 timed out. Robust software or HDL design should have defined behaviours in overload circumstances such as this (such as 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.

FPGAs can provide some semi-automatic detection and repair facilities. However, these approaches tend to rely on IP cores and may therefore increase the design footprint of the whole solution and thus also the justification overhead.

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: this is addressed in Section 6. Detection and protection of these errors is often a standard feature of FPGAs, but does not address the usual bit flip concerns familiar in microprocessor based designs.

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. Additionally, since FPGAs are reconfigurable, permanent failures due to electromigration or other causes may in theory be accommodated by reprogramming, although it is questionable whether dynamic reconfiguration of this kind would be compatible with the requirement to maintain a stable configuration.

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, it is rare to have access to the verification records. 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 assessment

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, certain types of FPGAs are vulnerable to spontaneous configuration changes owing to ionising radiation (see Section 6).

In this section we analyse the similarities and differences among the vulnerabilities found in microprocessor and FPGA based designs, and the implications this has for V&V requirements and available approaches in each case. We compare microprocessor-based *systems* and FPGA based *systems*, considering small and medium sized applications at Cat A. Such systems comprise the core FPGA or microprocessor chip, supporting electronics and proprietary module structure, such as hot swap boards in a single chassis, using a proprietary interconnection system and development tools. We do not consider systems that are housed in more than one chassis or heterogeneous systems. The extent of the theoretical vulnerabilities and strengths of microprocessors and FPGAs *chips* often diverges from what is achievable with the systems on the market that are suitable for use in nuclear power plants, which include some element of common platform code: where this contains elements that are intractable to analyse (such as microprocessor platforms that make more than very limited use of interrupts) this places a limit on the feasibility of a deterministic analysis for an entire implemented system consisting of a platform and an application.

FPGA vulnerabilities concern common patterns of error in rendering requirements specifications in HDL and the tools used to refine HDL code into a deployed FPGA, and can occur both in IP cores (see Section 5.5) and per-development code. Since IEC 62566 mandates that all HDL designs be fully synchronous, if maximum logic propagation times for combinatorial logic do not generate unsynthesisable timing constraints, FPGA-specific vulnerabilities can in principle be reduced to toolchain vulnerabilities. Similarly, the logic functions generated by synthesis tools are in principle correct by construction, although some cautions and provisos are discussed below. An analogous situation with microprocessor based designs is that some classes of memory interference bugs are extremely easy to introduce in manually written assembly code programs, but are completely avoided by use of a suitable high level programming language – but the protection is lost if the compiler is flawed or the tool flow not followed properly.

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 (see Figure 3 and Figure 4). 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" [6][3]. 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 clause 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 Clause 14.1.1.

Clause 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." 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 clauses 6.5 and 8.5) discusses the requirements for using "Electronic System Level" (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 [6] (at clause 8.5) raises a number of provisos if HDL is to be produced by an automated toolchain.

Both IEC 60880 and IEC 62566 are silent on the use of even higher level tools such as Simulink and Matlab and traceability through tools such as DOORS to system requirements and plant need, but such techniques are increasingly deployed in modern design flows, and can also apply to systems that are eventually implemented using microprocessors.

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 logic synthesis process from HDL to RTL 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

[3] Clause 3.4

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 can 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

At the application level, timing events may be asynchronous. For example, a demand on a safety alarm system may come at any time. Further, the state transitions as a result of these stimuli might be immediate, or constrained as a functional requirement to occur with some specific delay or within some time interval. However, this transition bears little relationship to the time it takes to compute the next state from the current state. If the next state function involved a computationally intensive mathematical function involved in modelling the state of a reactor, it cannot be taken for granted that the state transition time is negligible. This has two consequences at the level of HDL or ordinary programming languages:

- there is some implicit or explicit constraint on worst case execution time for a particular piece of code
- given a non-zero state transition time at the synchronous level, there are implicit constraints at the application level on the number, frequency, parameters and coincidence of demands

The application's implicit or explicit requirements for timeliness of response must be matched with worst case execution time of the implementing logic. In microprocessor systems, this is often fraught with practical problems, and empirical and statistical arguments are often used rather than analytical techniques to provide assurance of this correspondence. In FPGA designs, spatial separation of functions can make this kind of analysis easier. It should be noted that it does not remove it altogether, because pipelined functions, particularly where there are cyclic data flow structures, still require an element of worst case timing analysis. Further, combinatorial logic takes a minimum time to settle that is dependent on both logic synthesis and place-and-route. However, timing analysis at this level is simpler than the equivalent for a microprocessor because it bypasses many low level control details, assembly language, operating system and processor dependencies that need to be dealt with the in the microprocessor case. Table 4 summarises the equivalences between areas of concern at the synchronous code level.

	Imperative microprocessor programming	**FPGA-implemented design**
Transition	Assignment of expressions	Clocked combinatorial logic
Iteration	Looping and recursion	Cyclic data paths

Table 4: Transition and iteration analogues

Timing constraints can arise not only from external demands, but from implicit requirements of internal concurrent behaviour. If a particular application state transition A is dependent on some other transition B occurring before or after another transition C, subtle timing bugs can occur, leading to (potentially intermittent) opportunities for starvation.

Synchronous level languages such as ordinary C or synchronous subsets of HDL are not sufficient to completely (or deterministically) describe a system behaviour at the C or HDL programming level. This is because sources of asynchrony from the application and physical levels often undermine assumptions necessary for synchronous operation. We will see that this problem is more difficult to deal with in the microprocessor case than with HDL-based designs, owing to the presence of interrupts and operating systems, the constraints on which are usually incompletely defined. V&V for FPGA based systems can be somewhat more straightforward in this area.

5.2.1 Microprocessors

At the physical level, from the programmer's or compiler writer's point of view, a microprocessor is a discrete state machine. The state of the processor and its memory is a deterministic function of some starting state, a program, and some set of state transitions that are uniquely determined by instructions in that program. While this means that the programmer is protected from any concerns about gate design, propagation time or latching stability of integrated circuits, the apparent determinism of a microprocessor is illusory:

- Usually the microprocessor may be interrupted. Even if sources of interrupts external to a system are regimented so as to occur with some minimum interval, software generated interrupts are much more difficult to constrain, as the times at which they occur is highly dependent on the final compiled object code, the data on which a program is operating, and the discontinuity of running code caused by operating system pre-emption or other interrupts.
- Microprocessors may pipeline concurrent processes to optimise instruction throughput.
- Contention on buses (where two parts of a system want to assert data on a bus at the same time) is resolved by hardware bus arbitrators, which are metastable and hence non-deterministic.

5.2.2 FPGAs

On the other hand, a common design paradigm for an FPGA based development assigns dedicated hardware to particular functions, so the difficulties born of contention and scheduling in a microprocessor design do not apply. On an FPGA, a particular piece of logic connected to an input/output takes the place of an interrupt in dealing with asynchronous external processes. It is still necessary to consider the frequency of demand of these "interrupts" and how the input handling logic synchronises external demands with the clocking domains and the processing cycles of any iterating structures, but the overall complexity of these issues is reduced. The main drawback of FPGAs where physical timing is concerned is that the programmer and toolchain must handle low-level propagation, synchronisation and stability issues that a microprocessor programmer may safely neglect.

The tool chain used to program an FPGA is similar to a compiler for a microprocessor. However, it is more complex as it has to take account of physical factors on the IC (in logic synthesis and place and route), and also frequently involves guided refinement and higher level transformations than are the case for a microprocessor compiler. Timing can be complicated by "back annotations", where the transformation tool refines timing requirements that can have an effect at the RTL level, and may need some re-verification. For example, if a very long chain of combinatorial logic is included in the flows between some clocked registers, then it may not be synthesisable, owing to the cumulative propagation delay through the gates. This violates the assumption that, from a microprocessor user's point of view, compilation is an acyclic refinement process. These complications and others in the toolchain provide more scope for tooling-induced errors, and issues with initialisation, power loss and transients.

A further vulnerability exists if a Verilog or VHDL development is not confined to a synthesisable subset. Behavioural directions such as "wait 10ns" are not synthesisable, and can be ignored by synthesis tools. In a synchronous design, waiting should be enforced by the clocking of registers only.

A problem can exist even in a fully synchronous design if registers are clocked at a different frequency. Section 8.4.7 of IEC 62566 prescribes static timing analysis to deal with this kind of problem. Section 8.4.7 of IEC 62566 does not fully explain what kind of static timing analysis is envisaged: some kinds are in effect part of the ordinary synthesis flow, whereas others can be carried out using back annotations generated by the synthesis flow. Section 8.4.7.4 of IEC 62566 mentions that clock skew should be analysed. This is only possible in external tools for those parts of clock transmission that are handled in programmable logic, or which are made known by the FPGA manufacturer.

This can be contrasted with the microprocessor case, in which timing is only non-deterministic if interrupts are used, or if a very aggressive multithreaded pipeline is employed. We consider the use of asynchronous interrupts to be an application-level timing issue. WCET for functions implemented in microprocessors is a matter that belongs at the synchronous level of abstraction, since assembly code is no less time deterministic than C, and rather more detailed.

Avoidance of timing errors in synthesis flows is reliant on particular FPGAs and synthesis tools, so some mitigation of these vulnerabilities can be achieved by platform and tool diversification.

Static timing analysis of the kind mentioned in Section 4.2.2 is an essential verification activity but addresses low level integrity concerns rather than high level timing constraints. Any hardware level timing constraints on the system boundary should be incorporated into the design and it should be verified that the design meets them. That the timing requirements themselves are correct should be validated by desktop review and integration testing.

5.2.3 Comparison summary

In summary, timing vulnerabilities arise from

- the potential for mismatch between an application requirement for an output within a particular time and the length of time microprocessor code or an HDL design takes to produce it
- consequential effects on concurrency and coordination affecting either microprocessor code or HDL
- low level physical propagation, latching and clock domain issues, affecting only FPGAs

With the exception of FPGA low-level issues, the principles of V&V techniques for checking that application timing assumptions are not violated are the same across platform architectures. Suitable assertions over HDL (combined with automated formal analysis or simulation tools) can provide a similar effect to some kinds of WCET analysis, should require less work to verify, and provide greater determinism with more confidence. However, it is important to check that any such assertions are adequate to cover the application requirements, as HDL is not a natural language to express application level constraints.

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. In an FPGA design, 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, but mistakes 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 such as QAC 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, which is a type of behavioural attribute discussed in Section 4.1.

5.5 INCORPORATION OF THIRD PARTY CODE

Third party code can be found in both microprocessor and FPGA implementations. In both cases, the issue is whether the source code is available for the third party code, so that it can be analysed. In microprocessor implementations, some hidden code for missing instructions such as floating point operations can also be included as library functions, in which case object code verification is often necessary. Similarly, it is important to check whether any synthesis optimisation processes in an HDL design include any IP from hidden libraries. The key differences between third party code for microprocessors and FPGA is that it is in general much more difficult to obtain source code for FPGA IP cores, and to be sure that the synthesis does not cause unexpected interactions between IP blocks and bespoke code (spatial isolation on the final layout can help here). Source code access to FPGA IP cores is typically much more expensive than netlist access, and the complexity of analysing such designs is further complicated by the fact that many have architectures that are parameterisable with some higher level tool. To keep the cost of IP cores down, in other applications, encrypted netlists are often used, but these should never be used as they are not compatible with IEC 62566.

IEC 62566 does not prohibit the use of IP cores (there are provisions about their inclusion in Clauses 7 and 9.3), but compliance is difficult for the reasons above.

The use of IP cores should be avoided where possible. If an IP core must be used, it may be easier to justify the use of an IP core supplied by the FPGA manufacturer or one of their affiliates: such cores are more likely to have good supplier pedigree and a wide body of evidence of prior use.

Clause 7 of IEC 62566 specifies how the requirements for an IP core should be captured and the acceptance criteria for candidate cores, most of which involve review exercises that are similar in scope to the activities needed to justify other kinds of COTS components. IP cores are often supplied “pre-verified”. Clause 7.4.2.1 of IEC 62566 requires that the verification of a pre-developed block be reviewed as part of the acceptance process.

5.6 UNREVEALED IMPLICIT STATE CORRUPTION

Wherever there are cyclic structures in code for microprocessors or FPGAs, there is capacity for SEUs or other physical defects to cause a transient problem that causes internal state to diverge from what it should be (where a correct current state is defined as a function of an input history). It is easier for such failures to be revealed in a microprocessor design, because important state held in static memory can be checked by dedicated code, while local data on the stack is ephemeral. In the case of FPGAs, a corrupt state is harder to detect, as it is physically distributed, and there are no out-of-band facilities to detect application data corruption directly. (Application data is here distinct from configuration data for the FPGA, for which consistency checks against a stored configuration are often implemented by the FPGA hardware.) Figure 5 illustrates to process blocks A and B that are capable of accumulating state. If B is, for example, computing a rolling average of some quantity that is being sampled through the external input to A, any transient corruption will not be detectable, but will influence later calculations of the rolling average.

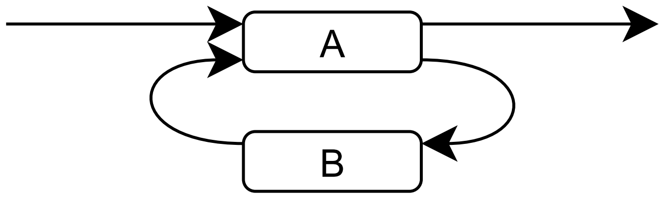


Figure 5: State loops

The use of cyclic structures should generally be minimised where possible. If a cyclic structure is being used as a hardware analogue of a "while loop", a simple mitigation is to ensure that cyclic structures are reinitialised each time the loop is invoked by a sequential pipeline. If the loop is calculating some rolling quantity, then the algorithm should be designed so that the influence of earlier values decays with time. If more active checks are needed, then a layer of dedicated logic is needed to detect and/or correct transient errors.

Control or data flow loops in a design can be easily identified using static analysis techniques: this applies equally to microprocessor and FPGA based implementations of an application. Once identified, different degrees of verification are necessary depending on the justification or mitigation of the loop, or the potential consequences for the application if a state corruption occurs.

5.7 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:

- Especially for the more specialist FPGA architectures (e.g. antifuse designs) a given chip IC 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.
- FPGAs have all the same EDA and fabrication vulnerabilities as microprocessors, but in addition 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 manufactures, and are highly complex. They therefore introduce an entirely new area of vulnerability when compared with microprocessors.

Specifically, at the implementation level, an FPGA-based application can be susceptible to logic propagation delays, clock distribution problems, latches that are not given enough time to latch, fan-out delays and logic replication synchronisation issues.

In the case of an FPGA, if a synchronous subset of an HDL is used to specify the low level logic of a system as specified by IEC 62566 (isolating the implementation from application level timing issues), and assuming that the synthesis has been set up to allow sufficient settling time to latch registers after each piece of combinatorial logic, then the problems of Section 5.7 can be reduced to the level of confidence in the FPGA toolchain, rather than errors by the implementer of the application. This toolchain is typically a closely guarded proprietary secret of the FPGA manufacturer, so additional confidence may need to be obtained by using diverse FPGA implementations of the same logic.

Designs should be restricted to synchronous-only HDL designs as specified by IEC 62566 (this does not in itself require that only directed acyclic graphs of gates between registers be used). This design rule bars some common structures such as asynchronous bus arbiters and ripple counters. The ensurance of preservation of timing properties from HDL, through logic synthesis and place-and-route to bitstream, is in the hands of a manufacturer's closed source tool, and thus independent static assurance of timing correctness is likely to be impossible. The pedigree of the FPGA/tool manufacturer and standard of production is of even higher importance than that of a compiler for a microprocessor, since no object code is available to analyse after compilation. Although the trust relationship in this case is similar to that involved in trusting the verification activities of a microprocessor manufacturer, the addition of a synthesis tool adds an additional layer of complexity over hardware synthesis that creates more opportunities for errors than in the microprocessor case. This problem can be mitigated by using diverse FPGAs and tools to synthesise their netlists and bitstreams.

In the case of a microprocessor, analysis of electronic timing issues is out of scope (implementation timing at the assembly code level is a matter of code or discrete model refinement correctness, as assembly language semantics are discrete).

For FPGAs, a number of techniques are available. As observed in Section 5.1, there is little scope for analysing FPGA synthesis tools themselves. However, integrity properties of HDL designs can be analysed, as 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
- formal (static) verification of assertions

Tools are available from independent software houses that perform these tasks. Some of their functionality (regarding constraints imposed by synthesis for a particular target architecture) has to be provided with the cooperation of the FPGA manufacturer.

5.8 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 its I/O 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: these should be avoided.

5.9 VULNERABILITIES – CONCLUSIONS

V&V techniques to address vulnerabilities in FPGA and microprocessor implementations differ more than those involved for behavioural attributes. This is because many behavioural properties concern the black box or system level functionality of an I&C application, whereas vulnerabilities tend to be particular opportunities for making mistakes that arise from different development paradigms, tools and implementation targets. Some of these mistakes are particular to the languages used, and both HDL and ordinary programming language code can be checked by automated tools for adherence to coding standards in order to check that certain common pitfalls or practices associated with subtler errors have been avoided. Many of the most intractable vulnerabilities in microprocessor based systems, such as lack of certainty about the impact of interrupts on task interaction and overall system performance are absent in FPGA implementation flows. However, care must be taken to mitigate the lack of transparency in the code artefacts that are eventually uploaded to the FPGA, as well as any hidden microprocessor-style development paradigms that might be used in and HDL design, of which HDL assertion checking techniques do not necessarily provide good coverage. The lack of transparency in code artefacts is caused by the complex synthesis toolchain involved in producing an FPGA design. Many of the tools are closed source, and the toolchain is lengthier than for the microprocessor compilation workflow.

A tabular presentation of this section can be found in Appendix C.

6 FPGA technology-specific issues

In this section we review vulnerabilities that are specific to particular FPGA types. There are three main defences to these vulnerabilities:

- The selection and justification of the particular technology given its installation environment. For example, an FPGA local to a sensor in an area with high levels of ionising radiation will tend to suggest antifuse or Flash types.
- Selection of defensive design rules such as configuration integrity checking and range checking, supported by V&V measures to check the consistent application of these design rules.
- Testing in a realistic environment or simulation of disturbances.

6.1 SRAM

SRAM types of FPGA use static RAM to configure the lookup tables (LUT) and interconnection sensitivities on the chip. At initialisation, these patterns are loaded from some persistent storage and configure the device. This type of FPGA is susceptible to single event upsets (SEUs) caused by free neutrons or alpha particles interacting with latches. FPGA manufacturers have developed architectures that monitor the configuration state of the FPGA using cyclic redundancy checks (CRCs) and compare it against that in persistent storage and make corrections if necessary. Clause 6.4 of IEC 62566 stipulates how these issues should be mitigated using defensive design techniques, but is not explicit about the extent to which a degree of systematic checking is necessary that the design rules have been met. Fault injection techniques have been suggested in the academic literature to test these behaviours dynamically, but IEC 62566 does not explicitly require them.

The SRAM type of FPGA Includes those with on-board Flash used to initialise SRAM. The main advantage of putting the Flash memory on the chip is that it makes it more difficult to intercept and copy a bitstream, but this is not the dominant consideration in a bespoke nuclear application, where the physical security of these components and the facilities for their manufacture and preparation should be high.

6.2 ANTIFUSE

Antifuse FPGAs are “one time programmable” (OTP) devices. They are a relatively niche product, produced by, for example, Microsemi. Once programmed, the connection topology of the FPGA is fixed, and does not need to be loaded on power-up. They are resistant to SEUs affecting configuration, but are still vulnerable to SEUs on application data, and, like any electronic device, they can suffer from hard faults. Any such hard faults may not be revealed in ordinary use. Defensive design is therefore still necessary, but need not encompass on-line configuration checking. Static analysis techniques may be used to check compliance with design rules, but would require bespoke or customised tooling.

6.3 FLASH

Flash-based FPGAs use Flash cells built into the gates that controlling LUT and interconnection configuration. They are less susceptible than SRAM types to radiation bit flips, but are more susceptible than antifuse types. This type is not the same as a SRAM FPGA with an on-chip Flash area (which is sometimes used for convenience and as a mechanism to protect the intellectual property on an FPGA). Again, devices of this type are available from, for example, Microsemi. The same techniques as discussed in Section 6.1 apply, but the quantitative aspects of any analysis (in terms of tolerable SEU rates) would be different (see also Section 4.6).

EEPROM can be used in an alternative to bulk Flash memory, but tends to have smaller capacity and less read/write cycle tolerance, so it is not usually employed in new applications.

6.4 MODULES, MEZZANINE CARDS, BACKPLANES AND COMMUNICATION BETWEEN CHASSIS OR RACKS

FPGAs are usually used in prefabricated modules, which form part of a wider system. A wider system could include a variety of Cots components using common standards such as mezzanine cards (e.g. AMC, FMC), ADC/DAC cards, backplanes (e.g. AdvancedTCA, VPX), and communications systems for digital communication between chassis and equipment racks (e.g. Ethernet, RapidIO). It is difficult to axiomatise a coherent model of an ad hoc collection of these kinds of products, so, for nuclear applications, a unified platform supported by a particular manufacturer is usually preferred, such as Radiy RadICS or Westinghouse's ALS platform. The V&V tools available for these platforms is specific to the manufacturer. However, higher level standards for systems development processes such as IEC 61513 are relevant here.

7 Conclusion

We have compared verification and validation (V&V) techniques for FPGA and microprocessor techniques by considering the requirements of the respective standards, a behaviour based analysis, and a survey of vulnerabilities that are commonly associated with particular technologies and design approaches. Overall, we have found few systemic differences, although V&V techniques at lower design and implementation levels do diverge somewhat.

In our analysis of standards we looked for differences in V&V requirements in relevant IEC standards at all stages of the development lifecycle, and found very few significant differences. In general, the relevant FPGA standard is less prescriptive about the specific documents that need to be produced, and in some cases it has clarified requirements in the equivalent microprocessor standard.

In examining the V&V techniques applicable to establish correct behaviours, we found that many are the same or similar. In particular, many of the system level dynamic testing approaches are identical. However, timing and concurrency require different approaches owing to the different physical design and abstraction level of the different architectures. FPGA tools are generally much more sophisticated in the V&V support they provide when compared to ordinary programming tools, but some of this additional complexity is necessary in order to compensate for the additional complexity of FPGAs.

V&V techniques to address vulnerabilities in FPGA and microprocessor implementations vary in more respects than the other aspects that we have considered. Many of the most intractable vulnerabilities in microprocessor based systems, such as lack of certainty about the impact of interrupts on task interaction and overall system performance, are absent in FPGA implementation flows. Operating systems are absent in FPGAs (although some platform code may be shared, it should not have the same problems with interrupts). However, care must be taken to mitigate the lack of transparency in the code artefacts that are eventually uploaded to the FPGA, and coverage of any data flow or control flow issues at a high level of abstraction should be reviewed, since HDL assertion checking techniques are not optimised for this.

In assessing a suite of V&V measures chosen for a particular implementation, FPGA workflows tend to be supported by more comprehensive toolsets, but it is important to review the whole set of V&V techniques used in any particular case to ensure that all abstraction levels are adequately covered, particularly if the resulting justification must interface with another case that has been developed or reviewed by engineers more familiar with microprocessor based V&V processes.

8 Glossary

Abbreviation	**Term**
ADC	Analogue digital converter
AMC	Advanced Mezzanine Card
COTS	Commercial off-the-shelf
DAC	Digital analogue converter
EDA	Electronic Design Automation
EMC	Electromagnetic compatibility
ESL	Electronic System Level
FMC	FPGA Mezzanine Card
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
HPD	HDL-programmed devices
I&C	Instrumentation and control
IEC	International Electrotechnical Commission
IC	Integrated circuit
I/O	Input/output
IP	Intellectual property
LUT	Look-up table
NPP	Nuclear power plant
OOR	Out of range
OTP	One time programmable
RAM	Random Access Memory
RTL	Register Transfer Level
SEU	Single event upset
SRAM	Static random access memory
TLM	Transaction Level Modelling
V&V	Verification and validation
WCET	Worst case execution time

9 Bibliography

1. S. Guerra, FPGAs in Safety Related I&C Applications, Adelard document ref. P/733/309/527 v1.0. Adelard project proposal, 11 June 2015.
2. IEC 61226. Nuclear power plants – Instrumentation and control important to safety – Classification of instrumentation and control functions. 2010.
3. P Bishop, R Bloomfield and S Guerra. *The future of goal-based assurance cases.* In Proceedings of Workshop on Assurance Cases. Supplemental Volume of the 2004 International Conference on Dependable Systems and Networks, pp. 390-395, Florence, Italy, June 2004.
4. Nuclear Power Plants – Instrumentation and Control Systems Important to Safety - IEC 60880, Edition 2, 2006.
5. Nuclear Power Plants – Instrumentation and Control Important to Safety – General Requirements for Systems, IEC 61513, Edition 2.0, 2011.
6. 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.

Appendix A: Tabular comparison

The following table identifies the major points of difference between the two standards at each lifecycle phase.

	IEC 60880	**IEC 62566**
Requirements	Does not explicitly require verification of the requirements specification	Requires verification (critical analysis) of the requirements specification
Design and Implementation	Fairly prescriptive in terms of what must be considered Design considerations listed; no explicit statement that additional design rules may be needed on a case by case basis Requires that translators and tools are thoroughly tested, as well as being qualified according to the standard Requires that sufficient detail be included in design documents, but no further guidance is provided Requires verification of intermediate design products Does not explicitly require static timing analysis to be performed	Strongly recommended design constraints identified Listed design rules explicitly identified as potentially applicable, but the decision must be made on a case by case basis and reflect latest knowledge Requires tools to be qualified according to the standard, but does not explicitly require thorough testing apart from this Greater specificity on what design considerations must be documented, at a minimum Does not require verification of intermediate design products, although does describe a formal review process to be performed at the end of this phase Requires static timing analysis to be performed
Verification	Pre-developed products to be assessed only against the requirements of this standard Adequacy of selection process for pre-developed products not explicitly required to be justified, nor their use in the wider system Explicit requirements placed on documentation, including listing of individual documentation items to be produced Automated code analysis permitted, but justification of manual input is not required Informative annex provides information on potential verification activities	Pre-developed products to be assessed against the rules of their suppliers, as well as against the requirements of this standard Adequacy of selection of pre-developed products to be justified, along with their use and their conformance with their component requirements specification Similar information required to be contained in documentation, but the type of document produced is not similarly constrained Tests must be fully automated and manual input justified Greater specificity in identifying the minimum verification activities to be performed, including a requirement to perform static verification activities
Software / HPD aspects of system integration	Verification software tools not explicitly require to be compliant	Verification software tools should be compliant with requirements of

	IEC 60880	**IEC 62566**
	with requirements of this standard on software tools for development	this standard on software tools for development
Software / HPD aspects of system validation	Equipment used for calibration should be demonstrated to be suited to the purpose of system validation Software tools used in validation to be documented as an item in the validation report	No explicit requirement that equipment used for calibration must be demonstrated to be suited to system validation No explicit requirement for validation software tools to be documented in the validation report, although this is a requirement of IEC 61513
Modification	No significant differences impacting V&V	No significant differences impacting V&V
Software tools for development	No significant differences, with the exception of HPD-specific constraints around logic synthesis	No significant differences, with IEC 62566 requiring conformance with IEC 60880, with the exception of microprocessor-specific constraints around compilers
Acceptance of pre-developed products	Explicit about the quality documentation needed, and the development process used for the pre-developed product Operating experience must have been obtained under similar conditions Operating experience can never completely replace documentation evaluation, and can only be used to compensate for specific named weaknesses in design	Requires only that a review is carried out on the available design and verification documents of the pre-developed product Operating experience must have been obtained under equivalent conditions Operating experience may be used to compensate for limited documentation weaknesses in reliability or design, but no specific weaknesses are named

Table 5: Tabular comparison

Appendix B: FPGA development flow

Artefact		Activity
Plant need		
	⇘ ⇙	Requirements capture
Requirements specification		
Application level modelling using ESL tool (sometimes)		
	⇘ ⇙	Design and coding process
HDL source code or schematic diagram Register-Transfer-Level description		
	⇘ ⇙	Implementation: Synthesis
Netlist (gate-level description)		
	⇘ ⇙	Implementation: Place and route
Bitstream (binary image to be loaded on to the FPGA)		
	⇘ ⇙	Instantiation on hardware
Observed execution		

Table 6: Tabular representation of the FPGA development process

Appendix C: Comparison of V&V vulnerabilities

Vulnerability	**Microprocessors**		**FPGAs**	
	Explanation	**Design pattern mitigations and V&V activities**	**Explanation**	**Design pattern mitigations and V&V activities**
Timing errors – application asynchronous timing	Timely response to external stimulus or preserving order of internal events	Testing, formal methods applied to high level programming language with suitable assertions, model checking	As microprocessors	Use design rules to check synchronous only design. Validate data and control flow propagation assertions against application requirements and use simulation and formal methods from FPGA toolchain to verify.
Timing errors – instruction sequences	Assembly language, clock speed, compiler choices, optimisations and inclusion of hidden library code affect execution time of atomic transitions in high level programming language.	WCET analysis	N/A	N/A
Timing errors – physical level issues	Correctness of microprocessor silicon design and interaction with external components	This is a matter for the microprocessor manufacturer. Direct access to V&V information is not usually available. Check interfaces with external buses for source of non-determinism.	Propagation time, transistor technology and metastable constructions	Avoid metastable constructions by using synchronous-only design, enforced by design rule checkers. Use back annotations and the FPGA vendor's static timing analysis tools to ensure that place-and-route does not place unsafe bounds on times during which signals are stable. Use simulation and on-chip testing.
Initialisation	Chip initialisation dealt with at a hardware level	No scope for V&V other than by manufacturer, which is not normally available. Check initialisation semantics understood and any assembly code used in bootstrap/initialisation/reset.	Reset lines must be able to return all logic to defined condition, which must correlate to the starting assumptions within the	Use FPGA toolchain to check. Check assertions adequately address application assumptions.

Vulnerability	**Microprocessors**		**FPGAs**	
	Explanation	**Design pattern mitigations and V&V activities**	**Explanation**	**Design pattern mitigations and V&V activities**
			semantics of the application.	
High level code bugs	Common error patterns	Use integrity static analysis tools to enforce design rules, find common mistakes and risky practices.	As microprocessors	Use design rule checkers and assertion generators in combination with simulators and HDL formal verification tools.
Incorporation of third party code	Included libraries, operating systems and compiler introduced assembly code routines	Object code analysis. Conduct ordinary V&V of source code where available. Limit use and complexity of operating systems.	IP cores and shared platform code	Avoid IP cores and verify source code. Comply with IEC 62566 provisions on pre-developed blocks.
Unrevealed implicit state corruption	Memory corruption	Check use of memory checking as well as value plausibility and consistency checks in mainline code. Consider effects of state divergence between divisions.	Cyclic structures and bulk memory may keep state that diverges undetectably during execution owing to SEUs or localised hard faults.	Review all cyclic structures for consequences of state divergence and introduce monitoring logic if necessary.
Silicon design errors	Manufacturer errors in hard silicon design	Usually not possible to assess directly manufacturer's hardware V&V.	As for microprocessors. Interaction between bitstream and silicon-level design proprietary information.	Not available. Manufacturer's post place and route analysis tools should be used. FPGA diversity may compensate.
Reconfiguration errors	Limited to unwanted firmware updates or corruption of code in RAM	Check defensive coding practices used (such as range checking, memory checking and use of watchdogs) and firmware configuration locked down.	SRAM (and to a lesser extent Flash) types of FPGA are susceptible to bit flips from SEUs.	Check use of on-line configuration checking facilities.

Table 7: Comparison of V&V vulnerabilities

Appendix D: The strategy triangle of justification

The strategy is property-based, vulnerability-aware and standards-informed. It is described by the safety justification triangle of [3]. Each of these different aspects of the strategy is discussed in the following sections.

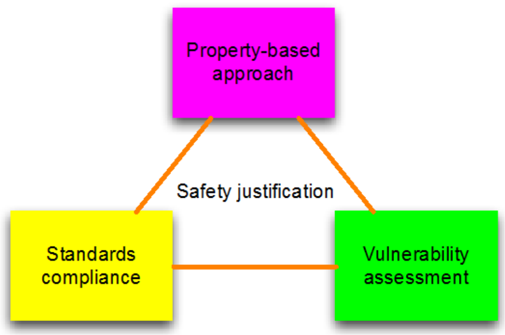


Figure 6: The strategy triangle of justification

D.1 Property-based approach

A property-based approach focuses directly on the behaviour of the system or component and explores claims about the satisfaction of the requirements and the mitigation of potential hazards. This approach is usually linked to specific claims about properties of the device being justified (e.g., time response, accuracy).

Different properties can be considered for different types of systems or components, and the approach is generally applicable to any I&C system.

D.2 Vulnerability-aware approach

Vulnerabilities are weaknesses in a system. They could lead to a hazardous situation (e.g., if a divide by zero is not caught by error handling) but are not strictly a hazard. Experience has shown that bad things can occur from them and so they should be considered within a vulnerability analysis viewpoint. Therefore, here we consider whether there may be vulnerabilities that would affect the ability of the device to exhibit the properties considered in Section D.1.

There are several methods and techniques that can be employed to perform a vulnerability analysis for a component and its system. Lessons learned from internal and external sources should be incorporated into the vulnerability assessment.

D.3 Standards compliance

Another principle, the third part of the assessment triangle, is that we should recognize the experience of others and, where there is a consensus, comply with appropriate standards.

The standards compliance argument would involve assessing the development process and design against relevant nuclear standards for a system performing Cat A functions: IEC 62566, “Development of HDL-programmed integrated circuits for systems performing Cat A functions”. For software-based systems, the corresponding standard is IEC 60880 [4].

VERIFICATION AND VALIDATION TECHNIQUES FOR I&C APPLICATIONS IN NORDIC NPPS

This report considers the verification and validation (V&V) techniques that can be applied to microprocessors and FPGAs. It finds that many techniques apply similarly to both, but at lower design and implementation levels the tools diverge, particularly when considering mitigations of vulnerabilities. Techniques for FPGA V&V are generally more comprehensive and integrated into the standard toolchains. Some of this complexity is needed to address extra design vulnerabilities present in FPGAs as compared to microprocessors, but in other areas the resulting analysis is arguably more routine and more thorough than is usually attempted for microprocessors. FPGAs are also free from some particularly difficult uncertainties and intractable analysis problems caused by the presence of operating systems in microprocessor-based platforms. Some behavioural V&V techniques dealing with application level issues such as data flow do not have such obvious analogues in HDL V&V methods based on hardware assertions. Consequentially, it is important to review the whole suite of V&V measures used for a given application to ensure that all abstraction levels are adequately covered, particularly if the resulting justification must interface with another assurance case that has been developed or reviewed by engineers more familiar with microprocessor based V&V processes.