



A Functional Verification Methodology for Highly Parametrizable, Continuously Operating Safety-Critical FPGA Designs: Applied to the CERN RadiatiOn Monitoring Electronics (CROME)

Katharina Ceesay-Seitz^(✉) , Hamza Boukabache, and Daniel Perrin 

CERN, European Organisation for Nuclear Research, 1211 Geneva 23, Meyrin, Switzerland

{katharina.ceesay-seitz, hamza.boukabache, daniel.perrin}@cern.ch
<https://home.cern/>

Abstract. Electronic systems that are related to human safety need to comply to strict international standards such as the IEC 61508. We present a functional verification methodology for highly parametrizable, continuously operating, safety-critical real-time systems implemented in FPGAs. It is compliant to IEC 61508 and extends it in several ways. We focus on independence between design and verification. Natural language properties and the functional coverage model build the connection between system safety requirements and verification results, providing forward and backward traceability. Our main verification method is Formal Property Verification (FPV), even for Safety Integrity Level 1 and 2. Further, we use constrained-random simulation in SystemVerilog with the Universal Verification Methodology and a design independent C reference model. When faults are discovered, the coverage model is extended to avoid regressions. Automation allows the reproduction of results and the reuse of verification code. We evaluate our methodology on a subset of the newly developed CERN RadiatiOn Monitoring Electronics (CROME). We present the challenges we faced and propose solutions. Although it is impossible to simulate the full design exhaustively, several formal properties have been fully proven. With FPV we found some safety-critical faults that would have been extremely hard to find in simulation.

Keywords: Functional verification · Safety · Formal Property Verification · Constrained-random simulation · Natural language properties · Functional coverage · Regression coverage

1 Introduction

When electronic systems are related to human safety, their whole life cycle needs to comply with strict domain-specific standards [1–4]. The general standard for

functional safety of electrical/electronic/programmable electronic safety-related systems is the IEC 61508 [5]. Its latest version dates back to 2010. The ISO 26262 automotive standard is the most modern one. Its latest version is from 2018 [1]. One very strict, but also quite old standard for safety-critical hardware is the DO-254 from 2000 [2]. The IEC 60532 standard for radiation protection instrumentation assigns Safety Integrity Levels (SILs) to certain radiation protection functions [3]. These SILs and corresponding requirements for the design and verification of safety-related electronic systems are defined in the IEC 61508 [5]. Since 2010, verification methodologies and electronic design automation tools for digital design verification have progressed at a rapid pace. Our methodology adds modern verification techniques to the IEC 61508's V-model flow.

The most common technique for functionally verifying the Hardware Description Language (HDL) code for Field Programmable Gate Arrays (FPGAs) is still simulation with directed tests [6]. Stimuli are applied to the inputs of the Design Under Verification (DUV) and the values at the outputs are examined. For highly parametrizable systems that have many input parameters of large bit-widths it would be extremely time consuming to manually specify all interesting combinations and calculate the expected output values. More flexible techniques like constrained-random simulation are available [7]. But even with this method it can be infeasible to simulate all possibilities for certain designs. Let's imagine we could simulate one combination at each CPU clock cycle of the workstation which executes the simulation tool. It would take roughly 146 years to simulate each of the possible input combinations of a single 64-bit vector. In reality, many CPU clock cycles will pass until the simulator applies a new stimulus. While in many cases it might be sufficient to simulate only representative values, it is often hard to tell which value ranges are representative enough to catch all corner cases. These might be rare inputs or combinations of extreme values of mathematical functions or boundary values [8]. Furthermore, for some input values it might be necessary to verify all possible combinations. Therefore, additional verification techniques need to be applied.

We propose a functional verification methodology that combines the state-of-the-art verification techniques of the semiconductor industry: Formal Property Verification (FPV) and constrained-random simulation using the Universal Verification Methodology (UVM) [9], both with functional and structural coverage collection, while complying to IEC 61508. We evaluate them for a highly parametrizable, continuously operating safety-critical real-time system. We propose a workflow that extends the verification process required by IEC 61508 with the following concepts (see also Table 1):

- Independence between design and verification engineers
- Semi-formal methods during verification planning and requirements review
- Formal methods as main verification method even for SIL 1 and SIL 2.
- Constrained-random inputs for (expanded) functional black-box testing
- Coverage for regression test cases
- Traceability from requirements over coverage model and Natural Language Properties (NLPs) to verification results and backwards
- Repeatability of the results

Section 2 summarises related work and background. Section 3 provides an overview of our methodology. Section 4 is an in-depth case study of applying our methodology to the CERN RadiatiOn Monitoring Electronics (CROME). Section 5 concludes the paper.

2 Related Work and Background

Independence between verification and design is very important in our methodology. Engineers are more likely to find faults in code written by other people than in their own [10]. The IEC 61508 does not mandate it, it only refers to application specific standards [5]. It is required e.g. by the DO-254 [2].

Formal Property Verification (FPV), also called assertion-based verification, can exhaustively proof that a property holds on a design. Many engineers still hesitate to use formal verification because of its perceived complexity [6, 11]. A campaign was launched at Intel to convince engineers of its benefits [11]. As we will also demonstrate, additional faults can be found with FPV in designs that had already been verified by simulation [11, 12]. Often it is only used for simple designs or control paths [8, 13]. In [14], each design was first classified as suitable or not for FPV. A design with our characteristics would not be suitable according to their criteria. Opposed to that we decided to use FPV as main verification method for a complex continuously operating safety-critical design and got indispensable results. Our methodology shows how to integrate it into a safety-standard compliant process.

Requirements-based testing is required by e.g. DO-254 [2] and ISO 26262 [1]. In [15], this method was extended with constrained-random simulation for robustness testing, or in IEC 61508 terms “expanded functional testing” [5]. Researchers in [11] mentioned the difficulty of tracking verification progress in FPV. We use the functional coverage model and Natural Language Properties (NLPs) [16] as connection between system safety requirements and test results. The methodology in [14] uses templates instead of NLPs that are automatically translated into SystemVerilog Assertions (SVA) by a proprietary tool. The importance of a consistent translation from properties in easily reviewable form to formal languages was also shown in [17]. The lack of such methods can lead to incorrect translations and additional iterations. In [18], each requirement was related to a test case and coverage model item. Encountered faults were added to a fault database, related to requirements and if necessary, the coverage model was extended. We call the extension of the coverage model “regression coverage”. Our approach (detailed in Sect. 3.2) was prior to that described in [19].

An advantage of FPV with SVA is that properties are proven directly on the HDL code. Several formal verification methodologies for FPGAs exist that require a translation from HDL to a formal model in a tool-specific language [13, 20]. To comply to safety standards, it would be necessary to derive this model from the HDL code [17]. Any used tools need to be qualified [5]. We decided for SVA, for which several qualified tools are available [17].

Constrained-random simulation is very useful for highly parametrizable systems. A large number of stimuli can be applied without the need to explicitly specifying them. Weighted constraints can be used to guide the randomization in order to increase chances of generating scenarios of interest while also testing unusual input combinations. This technique typically finds more faults than directed testing [8, 15]. SystemVerilog provides many features to ease the development of flexible testbenches as well as properties and sequences which can be used both in simulation and FPV [7]. The SystemVerilog UVM library facilitates abstraction into transactions and verification code reuse. UVM was released in 2011, after the publication of the IEC 61508. It has been standardized in 2017 [9].

It can be distinguished between functional and structural coverage. The functional coverage model states which scenarios are of verification interest. It is defined by the verification engineers. Structural coverage measures how many percentage of the Hardware Description Language (HDL) code have been covered [8]. A very effective coverage metric is Modified Condition/Decision Coverage (MC/DC), where each condition has to affect the condition outcome at least once. It is required by the DO-178 standard for software in avionics industry [4]. Simulation tools provide this metric as well for HDL code [15]. SystemVerilog covergroups, cover properties or assertions could also be added by the designers to ensure that simulation test benches cover important implementation details [8, 12]. This would not violate the concept of independence [12].

In this article we solely focus on functional verification. Measures for avoiding failures due to random hardware faults need to be considered additionally for any safety-critical design [5].

3 Our Functional Verification Methodology

IEC 61508 lists several techniques that can be chosen for verifying FPGAs. Table 1 lists the techniques that we chose plus some techniques that we added (A). The last 4 columns show the level of recommendation by IEC 61508 per SIL. Due to the large number of inputs (~ 200) of our design and our positive experience with FPV that we will highlight in later sections, we decided to use it as main verification method. We complement it with constrained-random simulation using the UVM. For software, the IEC 61508 requires traceability from system safety requirements to verification results and vice versa, as well as repeatability of the verification activities. We adopt these points for FPGAs.

3.1 Verification Planning

Our workflow, detailed in Fig. 1, starts based on the system safety requirements, design requirements and the specification. Consistency between the first and the last two needs to be verified [5]. Each verification requirements is related to at least one system requirement. If we encounter undocumented design decisions during verification, we report them first to the requirements engineers rather

Table 1. E/E/EP system verification requirements and techniques

Sections in IEC 61508:2010 - Part 2	Techniques	Required by			
		SIL 1	SIL 2	SIL 3	SIL 4
7.9.2.1-4 Verification planning	Semi-formal methods	A	A	A	A
7.9.2.5 Conformance to safety requirements	Requirements traceability	A	A	A	A
7.9.2.7, Table B.1 Verification of system design requirements	Inspection of specification	–	HR	HR	HR
	Semi-formal methods	R	R	HR	HR
7.9.2.8, Table B.2, Table B.5, Table F.2 Verification of the system design and development	Simulation	–	R	R	R
	Formal methods	–	–	R	R
	Functional testing on module level	HR	HR	HR	HR
	Expanded functional testing	–	HR	HR	HR
	Black-box testing	R	R	R	R
	Constrained-random input	A	A	A	A
	Coverage of the verification scenarios	R	R	HR	HR
	Coverage for regression testing	A	A	A	A
7.9.2.6, 7.9.2.10 Verification results documentation	Requirements traceability	A	A	A	A

A ... Additionally added to our methodology, - ... No recommendation for or against the method by the standard, R ... Recommended, HR ... Highly recommended

than the designers, to keep independence high. After consensus, requirements, specification and verification items are updated by the responsible persons.

In our methodology, the functional coverage model builds the connection between the verification requirements and the results. Each verification requirement needs to be described by at least one SystemVerilog covergroup or cover property for simulation or by at least one NLP [16]. Within each method the Mutually Exclusive and Collectively Exhaustive (MECE) principle should be followed [21]. For our kind of design we identified the following grouping inspired by [21] as useful: use cases, interesting scenarios, temporal relations, value ranges, stress tests, negated requirements. Input values should only be covered if they had an effect and verification passes [8].

The analysis of uncovered items might reveal internal design details. In order to keep independence high, we suggest that the design engineers should analyse the structural coverage reports and disclose as little information as possible to the verification engineers. The goal should be 100% functional and structural coverage. If it can not be reached, an analysis should be performed and it should be justified why less than 100% are acceptable [15].

Natural Language Properties (NLPs) [16] are our coverage model items for FPV. They are a semi-formal notation where natural language snippets are translated into SystemVerilog property snippets with a fixed N:1 mapping. E.g. one can use different natural language expressions to describe the same formal statement. “*Expr implies that Seq*” and “*Every time when Expr: Seq*” can be both translated into and implication “*Expr |->Seq*”. Technical details can be hidden by application specific NLPs, e.g. “*Cycle is the start of a measurement cycle*” is translated into “*(\$rose(mtValidxDI))*”. That way an unambiguous connection

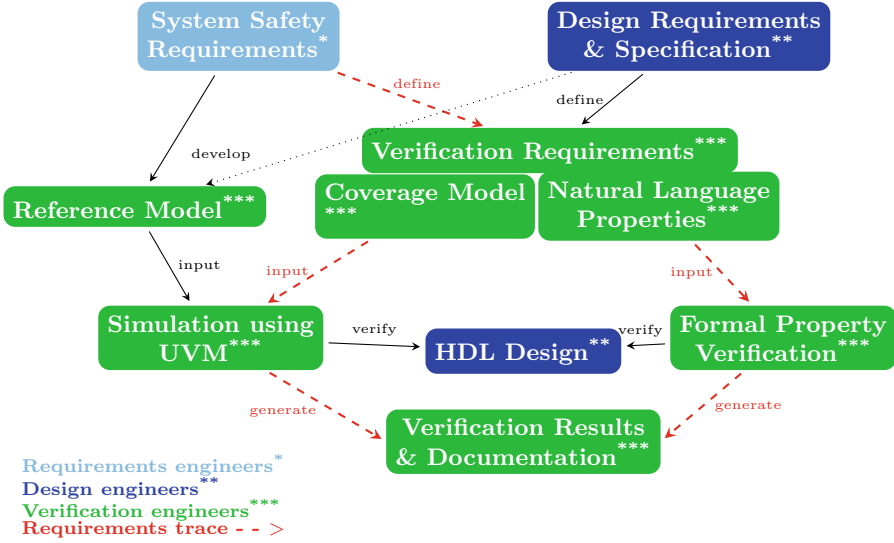


Fig. 1. Verification workflow

between a NLP and a formal property is established. The NLPs can be easily reviewed by requirements engineers unfamiliar with SVA. We use this review step to increase independence between design and verification.

For formal verification we calculate functional coverage as follows:

$$\text{Functional coverage}[\%] = \frac{\text{Nr. of proven properties}}{\text{Nr. of properties}} * 100.00 \quad (1)$$

Or with weights, similar to SystemVerilog coveragegroup coverage [7]:

$$\text{Weighted functional coverage}[\%] = \frac{\sum_i (w_i * p_i)}{\sum_i w_i} * 100.00 \quad (2)$$

w_i = weight per property, p_i = 1 if property was proven, 0 otherwise

3.2 Automated Verification

Formal Property Verification (FPV). Our main verification method is FPV with SVA [22]. We use as little formal assumptions, i.e. constraints, as possible, therefore even allowing scenarios that are outside the current specification. When the specification changes, the same properties can be reused for verification. This reduces the logic in the Cone-of-Influence of the formal properties, which makes it easier for formal tools to conclude. The properties can be validated in simulation by including the SystemVerilog file that contains the properties inside the SystemVerilog DUV interface.

The number of states of continuously operating designs grows exponentially with the number of input bits and necessary clock cycles for a proof. We model

complex calculations with 64-bit operands in auxiliary code and use properties to proof the equivalence of the DUV's outputs with the modelled calculations.

We start with black-box verification, which means that we do not modify or access internal signals and describe properties only in terms of input and output relations. If these properties are inconclusive, we apply abstraction techniques like inserting cut points on internal registers. That means that logic which drives these registers is cut away. This reduces the complexity of the proof calculation and therefore it is easier for the formal tool to conclude. A proof is valid for all values that are possible within the bit-widths of these registers. This includes the values that can be generated by the logic which is cut off. Therefore it is a logically safe transformation that might introduce false negatives, but never false positives [22]. Such techniques require the verification engineer to gain knowledge about the code, thus violating the principle of independence. Therefore, we apply these methods only after all independent black-box verification activities have been completed.

Constrained-Random Simulation. We use constrained-random simulation with the SystemVerilog UVM library whenever formal tools can't deliver results within reasonable time. Verification engineers develop a reference model based on the requirements, independent from the design engineers. SystemVerilog can communicate with a C or C++ model through its Direct Programming Interface (DPI). The test bench simultaneously sends UVM transaction to the Design Under Verification (DUV) and the reference model and compares the outputs.

Regression Coverage. Whenever a fault is found with simulation-based or formal verification that occurs in a scenario that is not yet part of the coverage model for simulation, we add it in the following way:

1. Identify input and output signal traces and their relationships that revealed the fault. Add a new covergroup, coverpoint or coverpoint bin. If a sequence of stimuli is needed to uniquely identify the scenario, use e.g. value transition or expression coverpoint bins or cover properties. Internal signals can be used if provided by the design engineer. We call these "regression covergroups/-bins".
2. Rerun the failing test and check that the new regression bin is covered in the same simulation time step in which the test failed.
3. Rerun all other test cases and check that the new coverage item is not covered by any other test that passes. If it is, the coverage item does not model a unique scenario. Either add step 6 or modify the coverage item and start at step 1.
4. The design engineer removes the fault of the DUV.
5. Rerun the failing test with the updated DUV. Check that the bin is still covered in the same simulation time step. Check that the verification passes.
6. Optionally: Copy the failing UVM test and UVM sequence class. Rename them to match the regression coverpoint bin and implement a stop condition. The test can stop when it has covered its corresponding bin. Add this test case to the regression test suite and do not modify it anymore.

Regarding point 3: In a continuously operating system, a unique scenario that leads to a fault might have to be described by long and complex signal traces and their relationships. Sometimes it can be more efficient in terms of engineering time to describe a signal relationship with a higher level of abstraction, which is not uniquely identifying the faulty scenario, but which includes it.

Regarding point 6: As long as the calls to randomisation functions in the regression test case are not altered, it can be used to reproduce the same scenario. SystemVerilog provides random stability as long as the order of new requests for random values is not altered within a thread [7].

Documentation, Traceability and Reproducibility. Forward tracing as shown by the arrows in Fig. 1 and backward tracing (by following the arrows in reversed direction) of verification items is used to measure verification progress and to provide verification evidence. All verification activities are documented in a version control system and can be reconstructed and reproduced.

4 Application to the CERN RadiatiOn Monitoring Electronics (CROME)

4.1 CERN RadiatiOn Monitoring Electronics (CROME)

CERN, the European Organisation for Nuclear Research, operates the world's most powerful particle accelerators. Particle collisions produce ionizing radiation. The radiation protection group is responsible for protecting humans from any unjustified radiation exposure. The CERN RadiatiOn Monitoring Electronics (CROME) are the new generation of instruments used for measuring ionizing radiation levels and triggering alarms and machine interlocks based on these measurements [23]. Several hundred units will be installed.

The CROME Measuring and Processing Unit consists of a radiation detector and an electronic system for data communication and storage, signal processing and safety-related decision taking. The latter contains a heterogeneous Zynq-7000 System-on-Chip (SoC) consisting of an ARM core and an FPGA. The ARM core executes an embedded Linux and an application that receives around 100 parameters with ranges up to 64 bit over the network, which it transfers to the FPGA. The FPGA performs the radiation dose and dose rate calculations. Based on that, it autonomously triggers alarms and machine interlocks. It contains all safety-critical code, implemented in VHDL. Triple Modular Redundancy and Soft Error Mitigation are used for detecting random hardware faults [24].

The devices can be used in areas with very different radiation levels, e.g. in service caverns close to the particle detectors as well as at the fences of the CERN site. To that end they were kept very generic and parametrizable. Periods of uninterrupted operation can last several months or years. These system attributes lead to high numbers of possible input values and deep internal states that are challenging for verification.

The *calculation of the radiation dose* consists of 3 additions with 64-bit operands, 1 multiplication with 32-bit operands and logic for rounding. It is calculated from the measured input current (fA - nA) and it is the base for one of the system's alarms. The dose is accumulated over a configurable period, which can last several years. Internal registers track the state. The calculation can be influenced at run-time by sending 6 parameters of up to 64 bit length from the CERN control room to a CROME device. The dose alarm decision is based on the outcome of 7 conditions, sampled on 2 real-time measurement cycles that can be thousands of clock cycles apart.

The *alarm and interlock matrix* block implements a complex configurable logical formula which drive the safety-critical outputs of the system. These outputs are connected to the alarm units, which provide visual and audible alarms and to the machine interlocks, which stop the particle accelerators in case of a too high radiation level. The formula can be configured by 200 2-bit wide parameters. In total the block has 2^{451} possible input values. Some of its outputs are fed back to the logical formula as input. Apart from that the block does not store an internal state and therefore results are available after a few clock cycles.

4.2 Verification Planning

We derived 76 verification requirements for the radiation dose calculation from only 26 system-level requirements, which were written with a very high level of abstraction and some ambiguities. The latter were discussed directly with the requirements engineers for increasing independence. 8 statements that were added and 10 statements that were only partially contained in the design specification lead to further verification requirements. The analysis lead to 12 updates of either requirements, specification or verification code. Verification planning lead to a more complete documentation of the whole project, which is very important to comply with safety standards [5]. We specified 52 cover properties and 4 covergroups that contained 56 coverpoints, as well as 30 NLPs.

During the review of the NLPs we discovered one very critical misunderstanding regarding the triggering of the radiation dose alarm. The design and verification engineers interpreted a requirement in the same way, but differently than the requirements engineers. This could not have been discovered with any automated verification technique. It shows the importance of independence and reviews. The detailed example has been reported in [16].

4.3 Automated Verification

Simulation and FPV were executed on a CentOS 7 workstation with 4 GHz CPU and 32 GB RAM. The single-threaded Questa Sim simulator, version 10.7, was used for simulation. Questa PropCheck, version 10.7c, was used for FPV with 8 hardware threads on 4 processor cores.

Formal Property Verification

Radiation Ambient Equivalent Dose Calculation. The dose calculation was modelled with auxiliary code. Properties compare the outputs of both models. So far, the dose calculation could be proven with the following constraints:

- 101 different calculation period lengths from 0 to 100, where 0 stands for an unlimited period
- Operands of additions restricted to 8 possible values or calculation period restricted to 2 real-time measurement cycles
- Time counting register restricted to 13.6 years in 100 ms unity, which therefore also limits the maximum period length to 13.6 years.

We allowed arbitrary values in the reset state by using a netlist constraint that sets the initial values of input ports to X. That means proofs cover every possible starting state, which includes the actual reset state.

A cutpoint was inserted at the register *etxDN* that normally loads the time counting register *etxDP* with a new calculated value. That means that the formal tool treats *etxDN* as an input and generates a proof for all possible values. If cutpoints are enabled, the auxiliary code also uses *etxDP*. That ensures that the formal tool uses the same value inside the DUV and the auxiliary model during one round of calculation. *etxDN* cannot be used in the model, as its value can be arbitrary in any clock cycles. The DUV and the model can perform their calculations in different clock cycles. The properties proof that, after a defined number of clock cycles following the start of one round, the outputs of both models are equivalent.

This way the unlimited number of consecutive calculations that keep track of an internal state is reduced to a smaller sequence of recurring operations. The elapsed time tracking register needs to be verified in a separate proof.

Proven: 8 properties could be fully proven, without any constraints on parameter values. Most importantly they include the proof of correct triggering of radiation dose alarms. The triggering is decided by 7 conditions at 2 consecutive real-time cycles with configurable distance.

Undocumented Design Decision Found: One fault happened only with very specific input bit combinations when an internal calculation result was negative and rounded. Even though thousands of inputs had already been simulated, this scenario had not been covered. To cover it, very tightly constrained simulation test cases were needed. The rounding mechanism was not documented. The coverage model had to be updated.

Fault that Happens After 7 years of Continuous Device Operation: In the division of the elapsed time value, one of the operands was treated as a signed value. The calculation was only wrong, when the most significant bit had value ‘1’. This fault could have never been found by black-box simulation because it would have required to simulate 7 years of device operation to discover it. For environmental radiation monitors it is a realistic scenario to operate continuously for such a long time. FPV revealed the fault within 1 s.

Alarm and Interlock Matrix. The logical formulas of the matrix were modelled with auxiliary code. Properties were used to prove the equivalence of the calculated values with the DUV's output.

Proven: The alarm and interlock matrix was fully proven with 46 properties.

Fault in Radiation Dose Alert: In one very specific input combination, the radiation dose alert was not triggered due to a wrongly specified range of a partially used VHDL vector. Many stimuli had already been simulated by the designer and user tests with the programmed FPGA had passed. Only FPV revealed the fault.

Output not in Safe State in Case of Invalid Inputs: The system requirements allowed 3 different values for certain inputs that were stored as 2 bits. The 4th possible value is illegal and not expected. No specific measures were implemented to handle that case, so the outputs would have been in inconsistent states and not in their safe state.

Constrained-Random Simulation

Table 2. Functional coverage of the radiation dose calculation

Cover type	Covered - all tests	Covered - passed tests	Nr. of coverpoints	Nr. of bins	Nr. of stimuli applied
Cover properties	100.00%	100.00%	–	–	16355
cgIntConditions	100.00%	93.98%	28	466	324647
cgIntRegression	100.00%	100.00%	3	3	250
cgIntValueRanges	91.95%	73.02%	17	656	249327
cgIntRobustness	7.15%	6.02%	8	392	280977
Total	79.82%	74.60%	56	1517	454200

Radiation Ambient Equivalent Dose Calculation. Table 2 shows the number of stimuli that were applied to reach ca. 80% of coverage with constrained-random inputs. The goal was not to find the minimum number of stimuli necessary to reach full coverage, but rather to simulate large numbers of stimuli in the proximity of interesting scenarios and corner cases in order to increase the chances of finding faults in operation conditions that have not even been considered.

A coverage bin can be a value, value range, value transition or a condition outcome. An additional condition for coverage sampling can be specified. E.g. sampling a value for radiation dose calculation period is only valid, when the whole period has been simulated. It is not meaningful to sample it already when it has been applied to the input. The period can span thousands of clock cycles or in real-time: days, months or years. Some faults only appear after a long sequence

of applied inputs and internal state changes, like e.g. the fault that would have happened after 7 years of operation that we found with formal verification.

We did not reach 100% functional coverage for all covergroups. This shows the shortcomings of simulation for continuously operating devices. The last two groups contain values and expressions that are related to the radiation dose calculation period. Since simulation is even slower than real time, it is impossible to simulate these scenarios with purely design independent black-box techniques. Code coverage confirmed that the only bits that were never toggled were the higher-order bits of registers that store time values. Toggle coverage reached only 78%. The rest was fully covered.

It is possible to access any internal signals from within the SystemVerilog testbench. The internal state, e.g. the elapsed time register, could be manipulated to simulate different real time values and reach full coverage. As discussed for cutpoints, to keep independence high, this technique should only be applied after independent black-box verification has reached its limits.

In a first attempt the simulation that created the coverage shown in Table 2 ran nearly 40 h. The cause for that long runtime were the cover properties. They contained many sequences that spanned over a large number of clock cycles, using SystemVerilog constructs like `##[1:$]`, which means that something happens after 1 or an arbitrary number of clock cycles. As long as a property or sequence is not yet covered, the simulator has to create a new instance of it at each clock cycle and check in each following clock cycle whether it has been covered. This construct is very useful to intersect different sequences at arbitrary times, but it comes with the cost of runtime increase. A more efficient alternative turned out to be covergroups that use expression coverage with value transition bins. The covergroup sampling did not add any significant overhead.

We tracked the test cases that actually contributed to the coverage of the cover properties. We ran each of them until its contribution to coverage of cover properties stagnated. Once all properties were covered, we executed the rest of the test suite with cover properties disabled. This approach led to a total simulation runtime of 3 h.

Table 3. Verification findings - radiation dose calculation

Found by	Update of specification	Update of implementation	Update of verification code	Total found by method
Review of requirements	4	5	6	9
Natural Language Properties	1	1	1	1
Review of design specification	1	2	0	2
Constrained-random simulation	5	9	2	15
Formal Property Verification	4	3	4	11
Total	15	20	13	38

4.4 Results Summary

Table 3 shows the faults that we found in the radiation dose calculation. Some findings caused updates of multiple artefacts. The total numbers per method should not be compared, because faults that had already been found with one verification technique had been removed and could have therefore not been found anymore with the other techniques. The results show that each described method contributes to the discovery of faults. Table 4 shows how long it would minimum take to apply each input combination that has been covered by formal proofs if we could simulate one stimulus at each CPU clock cycle. Many more simulation cycles would have to be added to cover all different traces of continuous operation, like e.g. different dose calculation periods. The fault that would have happened after 7 years of operation clearly shows this need. The fault that was not covered because it was hidden behind an undocumented design decision and the fault in the radiation dose alert show that testing only a few values per equivalent class is not always sufficient.

Table 4. Estimated minimum simulation time for the proven bits, compared to actual formal verification run time.

	Radiation dose calculation	Alarm & interlock matrix
Nr. input bits covered by proofs	70	451
Nr. proven properties	8	46
Estimated min. simulation time	9359 years	$7.99 * 10^{137}$ years
Runtime formal verification for all proofs	1.46 h	33 s

Documentation, Traceability and Reproducibility. We use the version control system Git to communicate design and verification artefact updates. It can be easily forgotten to update a version number inside the DUV or verification code. Git generates unique hashes for each commit. We used these hashes to track the faulty and updated versions, log files, planning and results documentation. Any state of the test bench can be checked out and results can be reproduced.

5 Conclusion and Future Work

We presented a functional verification methodology for highly parametrizable, continuously operating, safety-critical real-time systems implemented in FPGAs. The methodology can also be applied to digital Application Specific Integrated Circuits (ASICs). We started with a discussion of our methodology in comparison to the requirements of the IEC 61508 and related work. We defined a workflow that starts with the system safety requirements as input and verification results as output. Forward and backward traceability between these artefacts is provided via functional coverage items. We applied our technique of NLPs [16] to

aid the requirements review. Our main verification method is Formal Property Verification (FPV). This decision was supported by the discovery of several interesting faults and successful proofs. Additionally we apply constrained-random simulation with the UVM. For both methods we use functional and structural coverage as a metric for progress tracking.

The methodology was demonstrated on a subset of the CERN RadiatiOn Monitoring Electronics (CROME). We will further apply it to that system and use it for future FPGA or ASIC projects. There is still potential for further automation and usage of the UVM's concepts for reusability from block to FPGA system level. Fault injection will have to be added to address random hardware faults. Then we intend to extend the methodology to include SoC system level verification that also includes the software running on the ARM core. We are also working on unifying the reference model for simulation with the auxiliary code for formal verification to reduce effort.

References

1. ISO 26262: Road vehicles - Functional safety, International Organisation for Standardisation (ISO) (2018)
2. DO-254 - Design Assurance Guidance for Airborne Electronic Hardware, Radio Technical Commission for Aeronautics, Washington, USA (2011)
3. IEC 60532: Radiation protection instrumentation, International Electrotechnical Commission (IEC), Geneva, Switzerland (2010)
4. DO-178 - Software considerations in airborne systems and equipment certification, Radio Technical Commission for Aeronautics, Washington, USA (2000)
5. IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems. The International Electrotechnical Commission (2010)
6. Foster, H.D.: 2018 FPGA functional verification trends. In: Proceedings of 2018 19th International Workshop on Microprocessor and SOC Test, Security and Verification (MTV), Austin, TX, USA (2019). <https://doi.org/10.1109/MTV.2018.00018>
7. IEEE Std 1800–2017: IEEE Standard for SystemVerilog - Unified Hardware Design, Specification and Verification Language. IEEE Computer Society (2018)
8. Bergeron, J., Cerny, E., Nightingale, A.: Verification Methodology Manual for System Verilog. Springer, Boston (2005). <https://doi.org/10.1007/b135575>
9. IEEE Std 1800.2-2017: IEEE Standard for Universal Verification Methodology Language Reference Manual. IEEE (2017)
10. Arthur, J.D., Groner, M.K., Hayhurst, K.J., Holloway, C.M.: Evaluating the effectiveness of independent verification and validation. *Computer* **32**(10), 79–83 (1999). <https://doi.org/10.1109/2.796141>
11. Achutha KiranKumar, M.V., Bindumadhava, S.S., Abhijith Bharadwaj, A.: Making formal property verification mainstream: an intel graphics experience. In: Proceedings of Design and Verification Conference and Exhibition United States, DVCon United States, San Jose, CA, USA (2017)
12. Butka, B.: Advanced verification methods and safety critical hardware. In: 2012 Integrated Communications, Navigation and Surveillance Conference, Herndon, VA, pp. K3-1–K3-9. IEEE (2012)

13. Grimm, T., Lettnin, D., Huebner, M.: A survey on formal verification techniques for safety-critical systems-on-chip. *Electronics* **7**(6), 81 (2018)
14. Devarajegowda, K., Servadei, L., Han, Z., Werner, M., Ecker, W.: Formal verification methodology in an industrial setup. In: 22nd Euromicro Conference on Digital System Design (DSD), Greece, pp. 610–614. IEEE (2019). <https://doi.org/10.1109/DSD.2019.00094>
15. Butka, B.: Is the current DO-254 verification process adequate for the future? In: 2012 IEEE/AIAA 31st Digital Avionics Systems Conference (DASC), Williamsburg, VA, pp. 6A6-1-6A6-11. IEEE (2012). <https://doi.org/10.1109/DASC.2012.6382383>
16. Ceesay-Seitz, K., Boukabache, H., Perrin, D.: Semi-formal reformulation of requirements for formal property verification. In: Proceedings of Design and Verification Conference and Exhibition Europe, DVCon Europe, Munich (2019)
17. John, A.K., Bhattacharjee, A.K.: Qualification of hardware description language designs for safety critical applications in nuclear power plants. *IEEE Trans. Nucl. Sci.* **67**(3), 502–507 (2020). <https://doi.org/10.1109/TNS.2020.2972903>
18. Carter, H., Williams, P., Fitzpatrick, T.: Foliations of coverage: introducing functional coverage to DO-254 verification projects. In: Proceedings of 2019 IEEE Aerospace Conference, Big Sky, MT, USA (2019). <https://doi.org/10.1109/AERO.2019.8741814>
19. Ceesay-Seitz, K.: Automated verification of a system-on-chip for radiation protection fulfilling Safety Integrity Level 2. CERN-THESIS-2019-022, Geneva, CERN (2019). <https://cds.cern.ch/record/2672187/>
20. Jabeen, S., Srinivasan, S., Shuja, S.: Formal verification methodology for real-time field programmable gate array. *IET Comput. Digit. Tech.* **11**(5), 197–203 (2017)
21. Sprott, J., Marriott, P., Graham, M.: Navigating the functional coverage black hole: be more effective at functional coverage modeling. In: Proceedings of Design and Verification Conference and Exhibition United States, DVCon United States, San Jose, CA, USA (2015)
22. Seligman, E., Schubert, T., Achutha KiranKumar, M.: *Formal Verification an Essential Toolkit for Modern VLSI Design*. Elsevier Inc., Amsterdam (2015)
23. Boukabache, H., et al.: Towards a novel modular architecture for cern radiation monitoring. *Radiat. Prot. Dosimetry*. **173**(1–3), 240–244 (2016). <https://doi.org/10.1093/rpd/ncw308>
24. Toner, C., et al.: Fault resilient FPGA design for 28 nm ZYNQ system-on-chip based radiation monitoring system at CERN. *Microelectron. Reliab.* **100**, 113492 (2019). <https://doi.org/10.1016/j.microrel.2019.113492>