norden

Nordic Innovation Centre

NT TECHNICAL REPORT

TR 578
Approved 2005-05

GUIDELINE FOR DESIGN AND SAFETY VALIDATION OF SAFETY-CRITICAL FUNCTIONS REALIZED WITH HARDWARE DESCRIPTION LANGUAGE

Andreas Söderberg
Jacques Hérard
Lars Bo Mortensen

Nordic Innovation Centre
Stensberggata 25, NO-0170 OSLO
Telephone +47 47 61 44 00
Fax +47 22 56 55 65
info@nordicinnovation.net
www.nordicinnovation.net

ISSN 0283-7234

NORDTEST
a Nordic Innovation Centre brand

NT TECHN REPORT 578
Approved 2005-05

Authors:

Andreas Söderberg[1]
Jacques Hérard[1]
Lars Bo Mortensen[2]

Nordic Innovation Centre project number:

04056

Institution:

[1]SP, [2]DELTA

Title:

Guideline for Design and Safety Validation of Safety-Critical Functions Realized with Hardware Description Language

Abstract:

This technical report describes the very detailed design process, the verification process and the safety validation process of a safety-related electronic control system to be implemented mainly into an ASIC. The report aims to support the system designer team as well as the safety assessment team. In addition, this report is the fundamental basis of the Nordtest method "*Validation of safety-critical functions realized with hardware description language*" for conformity assessment of safety-related functions implemented partly in digital circuits.

Technical Group:

Safety and Reliability

ISSN:	Language:	Pages:
0283-7234	English	73 p. + 4 appendices

Key Words:

Functional safety, Safety life cycle, Safety validation, ASIC, FPGA, (V)HDL, IEC 61508, ECSS-Q-60-02

Distributed by:

Nordic Innovation Centre
Stensberggata 25
N-0170 Oslo
Norway

Report Internet address:

www.nordicinnovation.net

Contents

Preface

This report is the result of the Nordtest project: Guideline for Design and Safety Validation of Safety-Critical Functions Realized with Hardware Description Language (NTHDL), which was carried out during 2004. The work was done in co-operation between SP Swedish National Testing and Research Institute and DELTA Microelectronics.

The following persons have been involved in this project:

SP	Andreas Söderberg
	Jacques Hérard
DELTA	Lars Bo Mortenssen

The contribution of the SP group has consisted primarily in the provision of information about safety requirements from IEC 61508 and methods for safety validation. The contribution of DELTA has primarily been their acquired experience and the provision of information about ASIC design process and ASIC verification process.

Summary

This report serves as a guideline for designing, validating and verifying HDL-based safety functions. The report motivates the selected methods and motivates their conformity with specified safety requirements. The report primary addresses conformity validation against IEC 61508 restricted to the boundaries of custom defined circuits.

The first part of the report provides a brief introduction to the basic principles of custom defined integrated circuits designed using HDL, with a specific focus on the difference between an ASIC and a FPGA. This brief introduction is followed by a description of the design flow used for such circuits, which is a fundamental part of this report.

The next part gives an introduction to functional safety and its basic concepts. In this part, basic means and measures for achieving functional safety are described. Topics such as redundancy, applied fault model and behaviour at fault are discussed in order to give the reader a background for the principles addressed in the report.

The system design part is one of the two major parts of this report and merges the required design flow from IEC 61508 standard and the Space product assurance, ASIC/FPGA development, ECSS-Q-60-02 standard. This forms the basis of this report and hence provides the final argument of functional safety. The reader will find all documentation and the interlocking design phases in the design process with the corresponding procedures that are required from these standards within the scope of this report.

The next major part describes the process of safety validation and system verification of the resulting system produced by the required design process. This part also addresses all relevant requirements from the above mentioned standards. Commonly used and recommended methods for validation and verification are also described in this part. The part is ended by a flow graph used for showing how different analysis methods correlates and in which phase of the design certain methods are most suitable.

The report ends with four appendixes where the first appendix gives a very brief introduction of reliability theory. The second appendix contains a small VHDL-example of an hypothetic system used to describe certain issues in this report. The third appendix contains cross references between this report and IEC 61508 and finally the last appendix contains checklists intended to be used to support the application of this report in a real design.

Introduction

The main target of this project was to develop a guideline for validating HDL-based safety functions used in control systems for safety-related applications. The method will primarily concern the requirements of the IEC 61508 standard by extraction and reformulation of some requirements. The method for conformity assessment was developed concurrently with this report within the same project.

The guideline aims to support system designers and system safety assessors in activities primarily related to IEC 61508. Although requirements related to the scope of this method are fully addressed, this method does not claim to provide sufficient information to enable complete conformity with the standard in its full extent. The reader shall have all parts of IEC 61508 available when taking this guideline into practice.

There is a large difference between HDL defined functions and software defined functions. The HDL description is an alternative way of drawing a circuit diagram (i.e. defines hardware) while software defines a sequence to be performed by already existing hardware. On the other hand the HDL is developed to be as similar to a common programming language as possible.

Standardization and conventional methods for safety validation give poor or no support for the special features and properties of custom defined circuits, thus safety validation of systems containing such a technology might be difficult. Most methods used for hardware/software validation are not applicable without modifications for HDL.

This report aims to clearly describe which requirements from IEC 61508 should be fulfilled regarding safety functions realised with HDL and how these requirements should be interpreted for the special features and properties of this technology. The report does also function as a guideline for conformity assessment. The report includes an illustrative example of a HDL-based function.

1. References

[1] Space product assurance, ASIC/FPGA development, ECSS-Q-60-02, Draft Specification, Issue 4, Revision 3.

[2] IEC 61508-1 Functional Safety of electrical/electronic/programmable electronic safety-related systems – Part 1: General requirements

[3] IEC 61508-2: Functional Safety of electrical/electronic/programmable electronic safety-related systems – Part 2: Requirements for electrical/electronic/programmable electronic safety-relates systems

[4] IEC 61508-3 Functional Safety of electrical/electronic/programmable electronic safety-related systems – Part 3: Software requirements

[5] IEC 61508-4 Functional Safety of electrical/electronic/programmable electronic safety-related systems – Part 4: Definitions and abbreviations

[6] IEC 61508-6 Functional Safety of electrical/electronic/programmable electronic safety-related systems – Part 6: Guideline on the application of Part 2 and 3

[7] IEC 61508-7 Functional Safety of electrical/electronic/programmable electronic safety-related systems – Part 7: Overview of techniques and measures

[8] Functional safety A Straightforward Guide to IEC 61508 and Related Standards, David and Kenneth, ISBN 0-7506-5270-5

[9] Control Systems Safety Evaluation and Reliability, 2nd Edition, William M. Goble,

ISBN 1-55617-636-8

[10] VHDL för konstruktion, Stefan och Lennart, ISBN 91-44-47782-1

[11] MIL-HDBK-338b, 1 October 1998, Electronic Reliability Design Handbook

[12] IEC 812, Procedure for failure modes and effects analysis

[13] IEC 1025 Fault Tree Analysis (FTA)

[14] START The Applicability of Markov Analysis Methods to Reliability, Maintainability, and Safety, Volume 10, Number 2, RAC Reliability Analysis Center

[15] STSARCES Quantitative Analysis of Complex Electronic Systems using Fault Tree Analysis and Markov Modelling

[16] Machine Control Systems – Safety Life Cycle and Probability Guideline for System Validation

[17] IEC TR 62380 Reliability Data Handbook – Universal model for reliability prediction of electronic components, PCBs and equipment

[18] Property Specification Language Reference Manual, Version 1.01, Accellera

[19] Professional Verification, Paul Wilcox, Kluwer Academic Pubishers, 2004

[20] Programmable Logic Handbook PLDs, CPLDs, & FPGAs,Ashok K. Sharma

2. Scope

The development process and validation methods described in this report concern safety-related functions realized with programmable logic or custom defined integrated circuits (IC:s) using hardware description language (HDL). In order to perform the safety validation of such a system, the methods used have to consider all aspects of custom defined IC:s.

The contents of this report address the aspects of a safety-related control system which consider custom defined complex ICs and describe means to validate such a system (function) according to relevant safety requirements. These requirements are extracted from common safety standards for control systems such as IEC 61508. The result of a validation using the finalized method conforms to these standards within the scope of this technical report. Other technologies used in the same system (e.g. microprocessors and software) are covered by other validation methods.

The report regards the design process, the allocation of safety requirements, the minimum set of documentation required, functional safety requirements and different analysis techniques. An example of subjecting some methods on a HDL-function is also included.

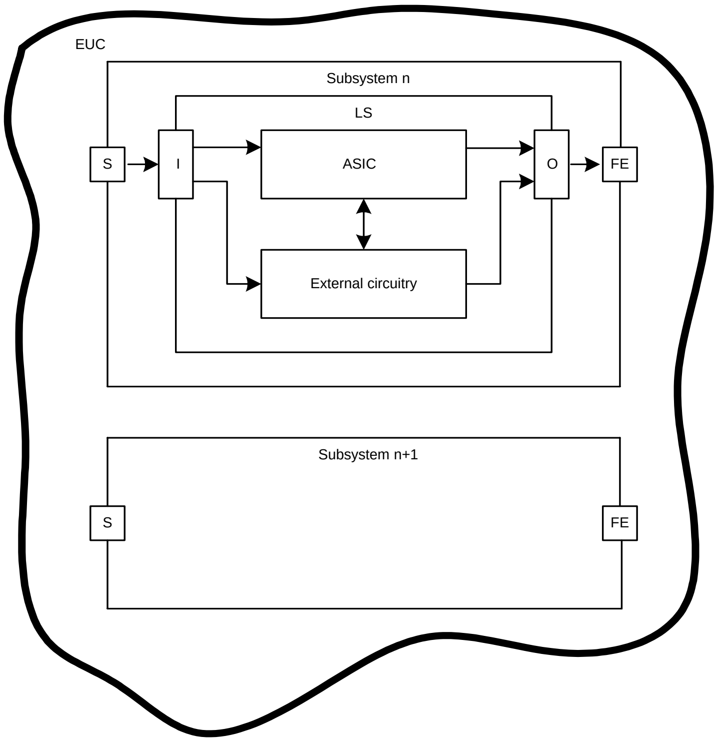


Figure 1 Application with an electronic control system.

Consider Figure 1 that illustrates n+1subsystems controlling the EUC. Each subsystem is equipped with sensors (S) and final elements (FE) connected to a logic system (LS). This connection is done through an input interface (I) and an output interface (O). This report concerns the input and output interfaces and a logic system when configured as shown in subsystem n. The EUC, the sensors and the final elements are excluded from the scope of this report. Any safety-related data communication between subsystems (e.g. communication protocols and safety principles) is also excluded from the scope of this report.

Regarding different system levels, the following denotation is used throughout this report:
-The E/E/PES design level describes how different subsystems are correlated.
-The subsystem design level describes how one of these subsystems is internally designed.
-The ASIC design level describes how the ASIC is internally designed.

3. Definitions and acronyms

Application Specific Integrated Circuit (*ASIC)*: A full custom or semi custom designed integrated monolithic integrated circuit that may be digital, analog or a mixed function for one user.
Ref. [1], section 3

Architectural Design review: *ADR*
Ref. [1], section 3

ASIC/FPGA Control Plan: *ACP*
Ref. [1], section 3

ASIC/FPGA Development Plan: *ADP*
Ref. [1], section 3

ASIC/FPGA Requirements Specification: *ARS*
Ref. [1], section 3

Critical Design Review: *CDR*
Ref. [1], section 3

Cell: Specific circuit function including digital and/or analog basic blocks
Ref. [1], section 3

Cell library: A collection of mutually compatible cells which conforms to a set of common constraints and standardized interfaces designed and characterized for a specific technology.
Ref. [1], section 3

Common cause failure (CCF): Failure, which is the result from one or more events, causing coincident failures of two or more separate channels in a multiple channel system, leading to system failure.
Ref. [5], clause 3.6.10

Data Sheet A Data Sheet contains detailed, functional, operational and parametric description of a component including block diagram, truth table, pin/signal description, environmental, electrical and performance parameters, tolerances, timing information, package description, and others.
Ref. [1], section 3

Design Validation Plan: *DVP*
Ref. [1], section 3

Design Validation Review: *DVR*
Ref. [1], section 3

Diagnostic coverage: Fractional decrease in the probability of dangerous hardware failures resulting from the operation of the automatic diagnostic tests

Equipment Under Control (EUC) equipment, machinery, apparatus or plant used for manufacturing, process, transportation, medical or other activities.

Electrical/Electronic/Programmable Electronic System (E/E/PES): System for control, protection or monitoring on one or more E/E/PE devices including all the elements of the system such as power supplies, sensors and actuators, input and output devices, and communication devices.

Failure rate (λ): Quantitative estimation of the probability for a part of a system or a component to fail in operation within a certain time period (e.g. per hour)

Floorplan: A floorplan is an abstracted, scaled layout drawing of the die, outlining the form, size and position of the major functional blocks and the pads including power7ground lines, clock distribution and interconnect channels.
Ref. [1], section 3

Hardware Description Language (HDL): is used to design the, or configure, in-circuit hardware (most commonly digital circuits) by the means of a programming language (e.g. HHDL, Verilog). The transformation between HDL and the circuit hardware is referred to as synthesis (compare to software compilation).

Initial Design Review: *IDR*
Ref. [1], section 3

In-System Programmable ISP**:** Term used to describe programming/reprogramming of devices directly on the printed circuit board, from a PC.
Ref. [20], section 8.5

Mode of operation: Way in which safety-related system is intended to be used, with respect to the frequency of demands made upon it, which may be either

> **low demand mode:** where the frequency of demands for operation made on a safety-related system is no greater than once per year and no greater than twice the proof-test frequency;
>
> **high demand or continuous mode:** where the frequency of demands for operation made on a safety-related system is greater than once per year and greater than twice the proof-check frequency

Ref. [5], clause 3.5.12

Netlist: A netlist is a formatted list of cells (basic circuits) and their interconnections
Ref. [1], section 3

Off The Shelf (OTS): Denotes components that is developed and manufactured by a third part relative to the design team, intended for use as a part of the design for which the internal functionality might not be known. The OTS component is often regarded as a black box component.

NOTE 1 The OTS component is often referred to as an IP-block (IP- Intellectual property) when designing using HDL.

Prototype device: A prototype device is a fabricated ASIC or programmed FPGA used to validate the new design in respect to functionality, performance, operation limits and compatibility with its system.
Ref. [1], section 3

RTL: Register Transfer Language

Safe failure fraction (SFF): Fraction of all potential failures that is safe failures. A safe failure does not have the potential to put the safety-related system in a hazardous or fail-to-function state.

Safety Integrity: The safety integrity of a safety-related system is the probability of the system to satisfactorily perform the required safety functions under all the stated conditions within a stated period of time.
Ref. [5], clause 3.5.2

Safety Integrity Level (SIL): Safety integrity level is a discrete level (one out of a possible four) for specifying the safety-integrity requirements of the safety functions to be allocated to the E/E/PE safety-related systems, where safety integrity level 4 has the highest level of safety integrity and safety integrity level 1 the lowest.
Ref. [5], clause 3.5.6

Safety lifecycle: The safety lifecycle refers to necessary activities involved in the implementation of safety-related systems, occurring during a period of time. The safety lifecycle starts at the concept phase of a project and finishes when all of the E/EP/PE safety-related systems, other technology safety-related systems and external risk reduction facilities are no longer available for use.
Ref. [5], clause 3.7.1

System on chip (SoC): a chip that holds all of the necessary hardware and electronic circuitry for a complete system. SoC includes on-chip memory (RAM and ROM), one or several microprocessors, peripheral interfaces, I/O logic control, data converters, and other components that comprise a complete computer system

Systematic failure: A systematic failure is a failure related in a deterministic way to a certain cause, which can only be eliminated by a modification of the design or of the manufacturing process, operational procedures, documentation or other relevant factors.
Ref. [5], clause 3.6.6

Test pattern A test pattern defines simulation stimuli and its expected responses (considering specific constraints to meet test equipment requirements) used to show correct behaviour of a device.

Validation: Confirmation by examination and provision of objective evidence that the particular requirements for a specific use are fulfilled.
Ref. [5], clause 3.8.2

Verification: Confirmation by examination and provision of objective evidence that the requirements have been fulfilled
Ref. [5], clause 3.8.1

4. Introduction to HDL Defined Integrated Circuits

With the introduction of standardized Hardware Description Languages (HDL) in the late 1980'es, the possibilities for defining Application Specific Integrated Circuit (ASIC) were greatly improved. Before, most integrated circuits were defined through configuration of cell primitives (AND-, OR-, Inverter-gates etc.). Simulation was mostly done through transistor level simulation and was very cumbersome. With the advent of HDL, digital circuits could suddenly be described on a behavioural level in a software-like manner, enabling easier simulation and debugging. It was however not until programs (synthesis tools) for translating the HDL description to hardware primitives became available in the beginning of the 1990'es that the use of HDL for system design, rather than just modeling, emerged. This suddenly enabled the designer to model the circuit behaviour and functionality, simulate it and translate it to an ASIC or an FPGA (Field Programmable Gate Array) that performs custom specialized functions.

The use of HDL transforms the electrical design process from being hardware design with components, PCBs and solder, to writing a software model that is treated by other specialized software (synthesis, router etc.) transforming the HDL model from pure software into an equivalent electrical hardware component. The transformation from HDL to hardware is controlled by the designer through software scripts, making the complete design realized through a sequence of software models and other software's interaction with the designer and the electrical design. As transformation progresses, the design becomes more and more attached to hardware, even though it still only exists as a software model of the intended electrical design. Figure 2 shows the HDL transformation to hardware through software interaction.

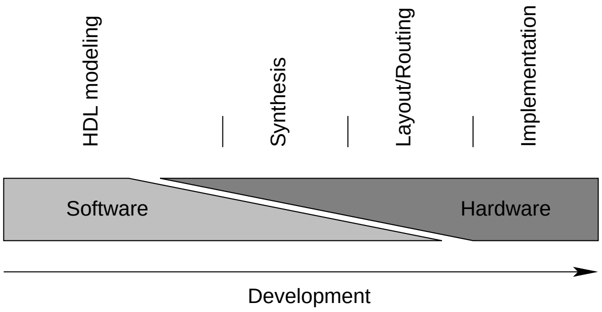


Figure 2: Software to hardware transformation.

Using HDL for defining customized components has several advantages over designing with standard components. First and foremost the designer gains control over the exact functionality of the HDL based component; this makes it possible to replace standard components of which only a portion of the functionality was utilized. Furthermore, the designer is able to select the exact implementation of particular functions that best suits the application, and not only what is on the market at the time. So if for instance redundancy for a specific function is required, it is only a matter of copying the HDL, whereas that specific function might not even be accessible for replication in a standard component.

HDL defined components make the system design less dependent of future component availabilities. In the case of an ASIC, choosing a common advanced technology, often guarantees component supply for many years, and thus a stable and robust system design. In the case of FPGA technology, the evolution often makes components obsolete in a matter of a few years. However, as HDL is highly portable, the HDL design can be remapped to a new FPGA family at an acceptable cost and with limited risks.

Development of HDL for ASICs and FPGAs is quite similar: the goal is to model the functionality in a way that can be translated to hardware (synthesized). From a synthesis perspective it is only a matter of choosing between different libraries with different primitives. The ASIC design consists of building the circuit all the way to transistor level and subsequently producing it, whereas the FPGA design consists of reprogramming a device. The verification by simulation of an ASIC is very demanding as it cannot just be programmed in an FPGA and tested. The ASIC workflow contains many more process steps to make the transformation to hardware and to verify that the transformation in each step was correct. Eventually the ASIC needs to be manufactured, this is a process which takes time and is quite expensive.

Table 1: Comparision of ASIC and FPGA technology

FPGA	**ASIC**
Expensive in low numbers	Very expensive in low numbers
Expensive in high numbers	Inexpensive in high numbers
Reprogrammable	Not reprogrammable
Cold start reprogramming necessary	No cold start reprogramming needed
High power consumption	Low power consumption
Short development time	Long development time

5. ASIC Design Flow

This chapter presents a description of a general ASIC (Application Specific Integrated Circuit) design development flow. The design development flow described here consists of 6 consecutive phases, taking the design of an ASIC from the early requirements specification all the way to the verified and validated prototypes ready for production. This description will mainly focus on digital design.

The ASIC development presented here, is based on the ESA ASIC/FPGA development specification [1], and shares the same development phases. Figure 3 gives an overview of the development flow.

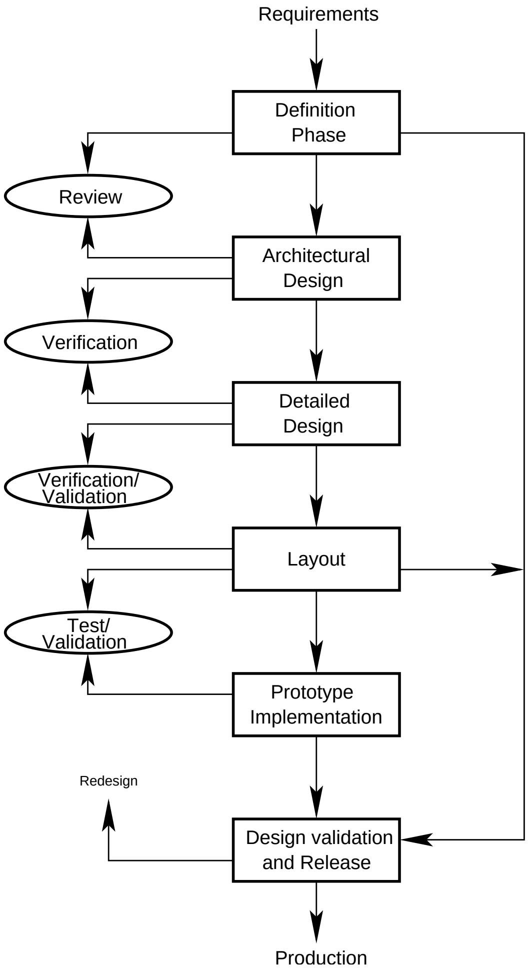


Figure 3: ASIC design development flow overview

The *Definition Phase* identifies the specific requirements for the ASIC. During this step, the architecture of the ASIC is investigated and planned, both on algorithmic and technological level. A suitable ASIC technology is identified and chosen. Feasibility study of chosen implementations is carried out and documented, justifying that the ASIC will meet the initial requirements. Risks assessment is carried out along with the feasibility study. The target of this phase is an *ASIC Requirements Specification* that will be used throughout the rest of the ASIC development.

In the *Architectural Design* phase the hierarchical design in terms of digital and analog breakdown is carried out. Based on the *ASIC Requirements Specification* the ASIC is designed on architectural level implementing the specified functionality and algorithms. Third party blocks (*IP, Intellectual Property*) is selected if needed, and verified before being integrated in the design. Along with the ASIC design work, a *Verification Plan* is established defining how the ASIC design is to be validated. The verification could also consist of prototype hardware implementation e.g. in an FPGA. Verification scenarios are defined and test simulations, called *test-benches,* are developed and run with the design or with design subblocks, verifying the design's correct functionality. Reviews are held to ensure that the design fulfills the requirements and that the test-benches sufficiently verify the functionality and possible error states. The output from this phase is a design model that can be simulated and where all critical blocks have been modeled sufficiently to define their exact implementation.

During the *Detailed Design* the design model is finalized. For digital design this involves that all modules and functions are made synthesizable (*RTL, Register Transfer Level*) and transformed to hardware primitive models (*gate level, logical hardware primitives*). The translation from RTL to gate level consists of synthesis, where the design is mapped to the chosen technology and optimized using the timing and electrical specifications in the *ASIC Requirements Specification*. Post production testing preparations are also made during this phase. Throughout the work in this phase the design is verified against the output from the *Architectural Design* phase through simulation or other verification methods. From the *Detailed Design* phase, the output from a digital design is a synthesized and verified model (*gate netlist*) ready for layout, and the output from an analog design is an electrical circuit netlist ready for layout.

The *Layout* phase is the next step in the transformation from RTL to hardware, where the gate netlist, consisting of logical hardware primitives is placed and routed; the netlist describes all interconnects between the gates, and the layout phase places all the gate instances on the chip area such that the interconnects can be routed correctly. Output from this phase is a full layer description of the ASIC chip implementation in silicon, including the content of the logical hardware primitives (transistors). Interconnect and gate delays are extracted from the layout implementation for timing verification and simulation with the design's test-benches.

The last step in the transformation is the *Prototype Implementation*, where the chips are manufactured in silicon according to the layout layer description. Each layer is transformed into a mask which is then used to transfer the layer definitions to the silicon chip into a photo lithographical like process. This step is usually carried out by a third party, called the *foundry*, since it requires a very specialized machinery and knowledge. After silicon production, the chips are packaged and are now ready for the use in the application.

The *Design Validation and Release* phase verifies that produced ASIC conforms with the specification both functionally, electrically and performance wise. It often includes field prototype testing of the ASIC in the application, but could also include characterization of the ASIC operation in various critical environments. After this step the ASIC can either go to production, or if failed to redesign.

In the next sections more details will be given about the specific tasks included in a general ASIC design development flow as described above. It will also include a brief description of the current tools and techniques employed during the ASIC design, and their purpose will be described shortly. Since there are several ways to establish an ASIC development flow, the project specific development flow is to be defined during the *Definition Phase,* according to [1]. The flow used here is an example of a representative design flow.

6. Introduction on Safety Requirements and ASIC design

This section introduces and describes some of the fundamental functional requirements applicable on safety-related E/E/PES derived from IEC 61508. This section also addresses how to interpret these requirements on the design and design process for an ASIC.

6.1. Introducing discussion

In IEC 61508 all integrated circuits used are generally considered as black-boxes. If an E/E/PES is primarily based on one large ASIC/SoC the designer/assessor has to focus on the very detailed internal design of the integrated circuit. However, there is no support for that in standards such as IEC 61508. The designer/assessor is instead confined to rely upon own interpretations. This sub-section is a brief discussion on fundamental safety requirements and provides some motivations on how to apply these requirements on an ASIC design.

In IEC 61508, the concept of redundant multi-channeled systems is handled where each channel is physically separated from each other. One channel is in most cases a ”stand alone” E/E/PES, often including one microprocessor with its sensors and final elements. Each channel has a PFH (Probability of Failure per Hour) that depends on the components used. When combining the channels the complete PFH for the entire control system (part of SIL requirement) may be obtained. This report addresses safety-related systems/functions implemented to some extent in one circuit (ASIC) combined with none or few external components, i.e. ”nearly” single channeled systems, using the terms of IEC 61508. See [16] for further guidance relating to control systems for machinery.

The SIL (Safety integrity Level) target is determined by a set of requirements related to the safety life cycle, described in IEC 61508-1. These requirements consider the complete design and operation of a device. The SIL is a measure of risk reduction. One part of the set of requirements addressed in this section concerns the probability of hardware failure.

When determining the level of PFH according to IEC 61508 some methods are commonly recommended. Examples of such methods are failure modes and effects analysis (FMEA), reliability block diagrams (RBD) and Markov Analysis. These methods are described in the following parts of this report. The PFH estimates the availability of the control system. The FMEA applies different failure modes on relevant components and examines their corresponding consequences. For an example, consider the BJT npn-transistor which has a limited set of failure modes.

Table 2: Example of an incomplete FMEA-table

No.	Component: BJT npn-transistor Failure mode	Failure effect	Comment	Probability of occurrence (λ)
1	Short c-b	Application related consequence	Any pin may be shorted to another on the silicon or on the PCB	A
2	Short b-e	”		A
3	Short e-c	”		A
4	Open c	”	Any pin may be opened on the silicon or on the PCB	B
5	Open b	”		B
6	Open e	”		B
7	h_{ie} altered +/- 50%	”	Functional failure	C

There are also other possible failure modes than those mentioned in Table 2 usually not considered without a specific application related reason. The letters A, B and C symbolizes failure rates for the actual component failure modes.

The probability of the occurrence of any failure in this particular component is referred to as λ (the failure rate). This parameter is often fetched from a data base or a component standard such as [17], and is defined by following equation: λ = 1/MTTF where MTTF is the Mean Time To Failure.

According to the standard, λ is assumed to be constant through the complete life cycle of the control system. The failure rate describes the probability of the occurrence of a *stochastic single-point failure,* implying that the failure occurs in the transistor and causes one of the transistor's failure modes after the predicted time MTTF. The PCB has to be considered as another component with its own failure rate.

The result of the integration of all the components in the system and the calculation of the final level of reliability (e.g. using reliability block diagrams) should be treated as an indicator of the approximate level of reliability. There are a huge amount of parameters influencing the reliability of each component and the assumption that the failure rate is constant through the complete lifetime of the system is a coarse approximation. Great care should be taken when calculating the confidence intervals for such a result.

One conclusion of this reasoning is that there are several levels of uncertainty in the process of determining the reliability (or behaviour at fault) of a system/function -
-It is up to the assessor to determine which components are related to the safety function
-It is the assessor who determines which failure modes are relevant for each component and how failure modes affect the functionality of the system both locally and globally.
-The estimated result of the calculations has a high level of uncertainty (one or two orders of magnitude). Hence the parameters that determine if the E/E/PES fulfils the functional safety requirements will, ultimately, depend on qualitative measures.

The PFHsys depends on additional parameters besides the failure rate of individual components. These parameters will also be described later in this report. Reliability data should always be treated carefully. It is recommended to only use the absolutely worst-case assumptions. This is natural in the qualitative approach since no one would like to rely upon misguiding or too optimistic reliability approximations on the adopted safety functions. The backside of that kind of reasoning is that usage of too pessimistic reliability estimations may bring injustice to a system that already has a sufficient hardware/software configuration causing overrating and hence increasing the level of complexity. A positive side of using the quantitative approach is that it forces the designer/assessor to regard the control system functionality from a different perspective when performing the analysis.

Another point of view of Table 2 addresses the nature of the failure modes used. Failure modes 1-6 can all be injected in the physical test object (i.e. it does not matter if they are occurring inside the transistor encapsulation or on the PCB) but failure mode no. 7 which is a functional failure can only occur inside the transistor by alternations in the silicon-die. Therefore the resulting failure effects may be explicitly determined.

When considering linear circuits (such as logical gate arrays, UARTs, buffers etc.) a way of reasoning similar to that applied for the transistor is used to establish the failure modes. Most of the failure modes concern shorts/open circuits between intersecting pins and simple functional failures. If the device has a somewhat more complex but still pre-defined functionality, methods such as FTA (Fault Tree Analysis) may be required to support the selection of functional failure modes. These components are classified as Type A components in IEC 61508.

As more complex circuits do not have a pre-defined functionality (such as microprocessors, digital signal processors, FPGAs etc.) the standard states that the methods mentioned above are insufficient for proving particular requirements. This is because these components functionality is defined by software that has to be analyzed; a specific failure rate with a reasonable confidence range can not easily be addressed. The only failure modes that are displayed in the FMEA are short and open circuits of intersecting circuit pins. Normal functional requirements on type B components consider the hardware configuration (N-channeled systems) and/or different techniques for monitoring the operation of the software that defines the functionality. Examples of these techniques are RAM/ROM checksum tests, software diversity, input/output plausibility testing, program flow monitoring etc. What is not mentioned in the standard is that these techniques are all really monitoring potential failure modes that may occur in the integrated circuit hardware. Such failure modes may be short-circuits in the in-circuit data bus, memory faults in the microprocessor program counter, faults in the comparators who evaluate conditional branches or fault in the instruction decoder look-up table (etc.) of a microprocessor.

Table 3 Integrated E/E/PES considerations

Aspects related to IEC 61508	Integrated E/E/PES
Type A components	The complete system is embedded in one circuit, due to the huge amount of type A components only groups of components composing one limited system function is considered in the assessment process (e.g. multiplexers, gates, minor groups of gates, adders, dividers etc.).
Type B components	All included Type B components are analyzed and treated as described in the standard (including software)
Hardware configuration	When the ASIC is mounted on the PCB its treated as a general off the shelf complex integrated circuit

Compliance with a SIL depends on several aspects including one or both of the following:

-The reliability of the function (i.e. of the components used)
-The system configuration (i.e. N-channel hardware configuration with physical separation)
In terms of IEC 61508, a SoC is always considered as a single-channeled E/E/PES regardless of integrated monitoring and redundancy.

Therefore in terms of IEC 61508 a single SoC carrying a safety function cannot alone comply with requirements on hardware fault tolerance (i.e. higher safety integrity levels) without any additional components that are used for providing functional and physical independence, see table 6 and 7. However, in systems where the safety function is highly dependent upon the ASIC functionality, the detailed design of the ASIC becomes a central part of the safety assessment. Even though physical redundancy is realized by a few and probably passive external components, the safety function will primarily depend upon the internal functionality of the ASIC. The main objective of this report is therefore to describe a methodology that maps relevant requirements in IEC 61508 on the design flow described in section 8 in this report.

By using techniques for increasing the level of fault tolerance in a SoC the reliability of that design will also be increased. In space technology several means and techniques for increasing the fault tolerance are commonly used in microprocessors. An example of such a technique is TMR (Triple Modular Redundancy) where, for example, the actual microprocessor consists of three separated processor cores placed on the same silicon-die and a 2oo3 voter. There are also techniques available for testing and evaluating the fault tolerance in a SoC. Examples of such a test is heavy-ion fault injection or soft saboteurs implemented into the VHDL model. What test technique to use depends on the applied fault model.

6.2. Generally about failure models

There are two types of failures to be considered: systematic failures and random hardware failures.

6.2.1. Systematic failures

A systematic failure is such a failure that is introduced in the design through an error, mistake or misunderstanding during a phase of the design process. A systematic failure is therefore related in a deterministic way to a certain cause and can only be eliminated by a modification of the design, manufacturing process, operational procedures, documentation or other relevant factors.

An example of a cause to a systematic failure is a latent firmware error that will manifest itself only under special circumstances and hence remains undetected during the integration tests. The fault will in this case be a part of the design and remain through the manufacturing process and thereby become a systematic failure.

6.2.2. Random hardware failures

The random hardware failure is caused by one or more degradation mechanisms types in the hardware. The failure occurs at a random time point. Such failures may be caused by faults of the types described in 5.2.2.1 – 5.2.2.3.

6.2.2.1. Transient hardware faults

The fault persists for a very short time, and then disappears. Example of cause for such a fault may be glitches arising from combinatorial logic where the output signals alter slightly different in time because of different delays in the logic network. Another example may be insufficient signal integrity (i.e. EMI related cause)

6.2.2.2. Intermittent hardware faults

The fault persists in a time-period and then disappears. An example of the cause of such a failure may be a transient failure occurring in the moment of altering the state in a state machine making one output erroneously low (or high) while the state machine remains in that state. It is although not likely that the fault appears the next time this state is reached.

6.2.2.3. Permanent hardware faults

Once the fault occurs it remains. Causes of such faults are always related to hardware degradation. Examples of causes of such faults may be component aging and/or wearing.

The event of a random permanent hardware failure may be expressed as a time dependent probability function and that is characterized by the failure rate. This is not possible with transient or intermittent faults.

IEC 61508 refers primary to permanent hardware failures and systematic failures in the requirements for hardware safety integrity. In [6] annex C an example is used to illustrate how to calculate the safe failure fraction of a subsystem. In the first step (a) the example assumes that a complex circuit has a distribution between the number of safe failure modes and the number of dangerous failure modes of 50%, because is not possible to analyze each failure mode.

According to IEC 61508 an ASIC is to be considered as a black-box complex circuit with a defined failure rate. If a safety function is almost completely integrated into an ASIC the designer should also regard transient and intermittent faults as well. Design principles and techniques for analysis and test may be integrated in the safety life cycle of IEC 61508.

6.3. Brief description of basic principles of redundancy

The primary purpose of using redundancy is to increase the availability of a system. The second purpose is to increase the robustness of the system. A redundant function is realized by duplicating an already existing function. Redundancy may be implemented at different system levels, depending on the requirements to be met. Often these original safety-related functions are very complex and hence expensive to fully duplicate. It is therefore common to only duplicate a specific part of such a function. All examples mentioned in the text below are only intended to explain/clarify the actual topic considered.

6.3.1. Hardware redundancy

By hardware redundancy is understood the duplication of parts of the electronic hardware in such a way, that if one part fails to operate, at least one remaining part will still deliver the correct service. Examples of hardware redundancy may be:

-N-modular-redundancy (nMR) where n denotes the number of microprocessors (channels) that are connected in parallel to a majority voter.
-Watchdog-timer: Device equipped with an oscillator (separated from the microprocessor oscillator) that monitors of the execution process in the microprocessor.

6.3.2. Software redundancy

By software redundancy is understood the duplication of parts of the software, so that if one part of the software fails to operate at least one remaining part will deliver the correct service.
It is possible to introduce redundancy at several system levels in software such as using several variables for critical data storage. Examples of software redundancy may be:

-Additional redundant conditions before initiation of a critical event in conjunction with special bit patterns (instead of single bits) for critical flags.
-Implementation of different redundant data paths between critical inputs and their corresponding outputs.

6.3.3. Information redundancy

By information redundancy is understood the duplication of critical information so that if one part of the information becomes distorted it remains at least one part that contains the correct information. Examples of information redundancy may be:

-Multiple storage of any type of critical information.
-One common means to duplicate information is to represent the actual information with a checksum. When increasing the uniqueness of the checksum, the degree of redundancy between the checksum and the actual information increases.

6.3.4. Time redundancy

By time redundancy is understood the duplication of the same critical control at several locations in a control sequence (series configuration). Each state in the sequence is redundant but the control sequence of outputs does not change until all states have been passed.
Example of time redundancy may be:

-Increasing the stability, by reading an analogue value using an ATD-converter at different points in time and then “comparing” these readings by making an average value.
-Duplicating a control module to a series configuration, assuring that the outputs remains correct through a sequence.

-Repeated transmissions of the same message via a communication channel.

6.3.5. Diversification

When using redundant functions, the system outputs are always, to some extent, dependent on the corresponding system inputs and hence redundant functions are more or less dependent on each other. Take for an example two relays connected in parallel and controlled by two redundant channels. If the relays are equivalent and delivered from the same distributor and manufactured at the same factory at the same time, a fault occurring in one relay will most probably also occur in the other relay at roughly the same time.

Such a failure caused by two simultaneous faults is denoted a common cause failure (CCF) and the risk of CCF arising because both relays are operated in similar environmental conditions, submitted to similar operational and electrical loads. The risk for CCF is relevant for all kinds of redundant functions and has to be considered. In order to reduce the risk of CCF the principle of diversification may be used. Diversification of a function means that a redundant function is designed and implemented by means different from the original function, but both functions still having the same outcome.

Assuming that the previously mentioned relays were controlling a motor, if one of the relays is exchanged by an electromechanical motor brake system so that one channel disables the electrical power supply to the motor and the other channel mechanically brakes the motor rotation, these two functions have become diversified. Another example of diversification is when using two redundant bytes in a critical software control flow and always keeping them inverted towards each other.

Diversification may be applied on all kinds of redundant functions, and the decision on whether to diversify a redundant function and the degree of diversification should be based on a CCF risk estimation. In IEC 61508 such estimation is expressed by the β–factor.

6.4. Diagnostics

When using redundant functions, the availability of a system is increased (i.e. the probability of a system to operate according to the specification at a certain time) but under a period during the product lifetime, the system will fail to operate regardless of implemented redundancy/diversification (unless repaired). It is therefore common to adopt mechanisms for diagnosing the system in order to detect if the system operates out of its specification.

Diagnostic mechanisms are usually designed so, that the result of the diagnostic test becomes input parameters to subsystems that affect the system operation. Such diagnostic mechanisms take responsibility for maintaining the correct system operation and, hence become an important part of the control sequence, instead of solely diagnosing the system. Such diagnostic functions should be considered in the system requirements specification.

When handling safety-related systems, it is common to add redundancy and diversity in conjunction with diagnostic functions, to ensure that critical failures are detected and handled in time to prevent the loss of safety-related functions.

Figure 4 illustrates an example of different operational states of a system and exemplifies the integration of diagnostic functionality in electronic control systems. The little circles containing numbers denote operational transitions.

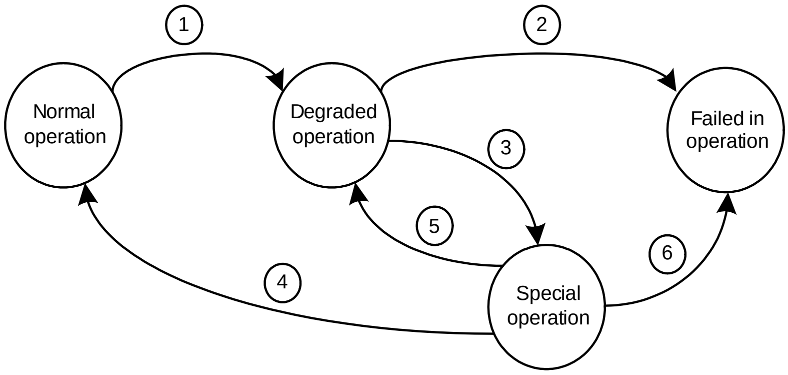


Figure 4: Diagnostic impact example on different system operational states.

The following text describes some aspects considered in Figure 4. This is only an example and gives a point of view for a reasoning that should be carried out in more detail for any system during design or validation that uses diagnostic functions.

Transition 1: A fault occurs and the system transits from normal mode of operation to a degraded mode of operation. The degraded mode of operation is very unpredictable, because the system operates apparently according to the specification but carrying a latent fault. This latent fault may manifest itself randomly (e.g. due to time of execution, certain combinations of input parameters or only in combination with an additional fault)
Example: An undetected fault occurring in one out of two redundant functions only affect the system service when the other function ceases to provide the correct service.

Transition 2: The fault prevents the system to make the transition from the degraded mode of operation to the failed mode of operation where it is no longer able to deliver its specified service (e.g. loss of the safety function). For some types of faults, the period under which the system remains in the degraded mode of operation may be negligible which increases the difficulty of detection.

Transition 3: The diagnostic functions implemented, provide the subsystems with information about the fault occurrence and hence forcing the entire system to transit to a special mode of operation where it remains delivering its service according to the specification but with reduced functionality (e.g. fail-safe mode).

Transition 4: The fault(s) become restored (e.g. repaired or the disturbance disappears) whereby the system transits from the special mode of operation back to normal mode of operation.

Transition 5: While the system remains in the special mode of operation an additional fault occur not detected by the diagnostic functions. This forces the system back to the degraded mode of operation where it will remain until an additional fault occurs and becomes detected (transition 3) or worse, undetected (transition 2).

Transition 6: While the system remains in special mode of operation, an additional fault occurs, causing the system to transit directly to the failed mode of operation (e.g. writing an erroneous value to the outputs that are assumed to provide some well-defined value in the fail safe mode).

7. System design

When designing safety-related control systems a similar design flow is used as when designing ordinary control systems but a number of additional items also have to be considered. During the definition phase the purpose of the final product is defined (concept) and possible limitations in usage have to be sorted out (requirements specification). The main objective when designing/validating a not-safety-related system is to prove that all system functionality is implemented according to the system specification. When designing/validating a safety-related system the main objective is to prove that the system ability of risk reduction is sufficient according to a risk analysis (which also incorporates the above). The difference between these two objectives is that the design/validation of a safety-related system always includes the target application and the design process performance. The system design part of this report covers two major areas, the documentation and the development process.

There are three measures that shall be considered when judging the achieved level of risk reduction (SIL):
Hardware safety integrity level – A quantitative measure that is primary based on the system behaviour at fault due to dangerous hardware failures. This measure is primary determined through design as described in this report and expressed as the PFH or PFD.

Systematic safety integrity level – A qualitative measure based on the applied methods and design principles for limiting the occurrence of systematic failures in the resulting product.
These methods shall control failures:

- caused by hardware and HDL design (i.e. implemented techniques and principles such as hardware diversity and diagnosis and control flow sequence monitoring etc.)
- due to Environmental stresses (i.e. implemented techniques and principles such as temperature monitoring, transient protection, over-dimensioning, separation between power lines and data signals etc.)
- during operation (i.e. implemented techniques and principles such as modification protection)

HDL safety integrity level – A qualitative measure based on the applied methods and techniques for assuring that the HDL code in an electronic control system achieves its specified safety functions under all specified conditions within the stated lifetime. The HDL safety integrity corresponds to the software safety integrity described in IEC 61508 wherever applicable. The requirements in this report are assumed to be sufficient (due to the design process and the verification process and the safety validation process) in order to fully conform to [4].

7.1. Documentation process

7.1.1. Introduction

The IEC 61508 standard sets out a generic approach for all safety lifecycle activities related to the development of electrical/electronic/programmable electronic systems (E/E/PES) that perform safety functions. The guidance and support provided by the IEC 61508 standard is intended for the development of systems based on commercially accessible electrical/electronic components.

The purpose of the present work is to define the complete set of requirements addressing E/E/PES that integrate custom designed components such as ASIC/FPGA.

7.1.2. Design documentation

Development of custom designed components is regulated by sector specific standards from which requirements are extracted. The requirements specification adapted to ASIC - based E/E/PES development is illustrated by Figure 5.

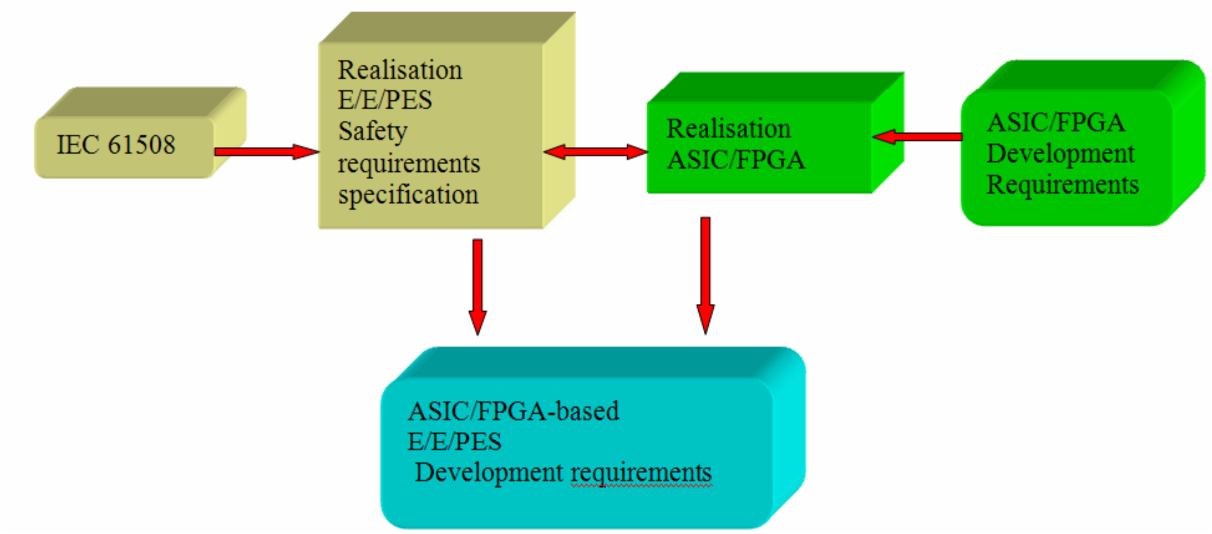

IEC 61508
Realisation E/E/PES Safety requirements specification
Realisation ASIC/FPGA
ASIC/FPGA Development Requirements
ASIC/FPGA-based E/E/PES Development requirements

Figure 5: Relation between safety requirements and design requirements.

7.1.2.1. Objectives

The first objective of the documentation requirements is to specify the necessary information to be provided to ensure that all activities necessary for the realization phase of the overall and software safety lifecycles can be effectively performed.
The second objective of the documentation requirements is to specify the necessary information to be provided in order to perform:

- the management of functional safety
- the verification
- the functional safety assessment

7.1.2.2. General

The E/E/PES is made up from a number of identifiable and separate subsystems. An ASIC is a complex component or a subsystem that together with the other subsystems shall implement the specified safety functions. The E/E/PE safety-related system design documentation shall specify the techniques and measures during the E/E/PES lifecycle phases to achieve the safety integrity level. In the present context only the Realization phase is of concern as shown in Figure 7 of chapter 8 System design.

The requirements that are listed in the following are retained under the assumption that other parts and subsystems of the E/E/PES compel to the requirements stated for the applicable phases of the IEC 61508 standard. As far as the ASIC is concerned, its realization is submitted both to the requirements of IEC 61508 concerning the realization phase as well as the requirements of the sector specific development requirements. Our first concern for safety development aspects, has lead our choice for support documentation to: [1] Draft Specification "Space product assurance – ASIC/FPGA development".

7.1.3. Documentation requirements

The requirements will be grouped according to the development phases of the ASIC/FPGA as it is described in [1].

7.1.3.1. Definition phase

All system configurations and characteristics, all issues imposing requirements on the device shall be collected in a definition status. The settled definition status shall have no ambiguity and shall cover all necessary resources for the design activities. The documents associated with that phase shall contain:

a) ASIC/FPGA Requirements Specification (ARS)
This document shall cover the items listed in [1], paragraph 5.2.2 a – s.

b) Feasibility and Risk analysis Report (FRR)
This report shall provide a judgement on the feasibility of the ASIC/FPGA development as defined by the Requirements Specification, as well as an assessment of the risks involved. This document shall cover the tasks listed in [1], paragraphs 5.2.3.1 and 5.2.3.2

c) ASIC/FPGA Development Plan (ADP)
This document shall identify the project external interfaces and constraints, the design flow, resources (equipment, software and personnel), the allocation of responsibilities, outputs to be produced and, finally, a schedule with milestones, decision points, type and number of design reviews.

d) Minutes of meeting (MoM) of the Initial Design Review (IDR)
The MoM of the IDR shall be added to the Management Documentation. During the IDR meeting the ARS, the ASIC/FPGA Requirements specification and the FRR are assessed. The MoM shall in case of a satisfactory result of the IDR meeting provide the evidence that an authorization to proceed with the Architectural Design has been given.

7.1.3.2. Architectural design

a) Architecture Definition Report
The Architecture Definition Report shall include the architecture broken down to the selected blocks, their interfaces, functionality/algorithms and interactions. A detailed text specification shall be edited.

b) Verification plan
The Verification plan shall define how the functionality and non-functional requirements stated in the Definition Phase Documentation shall be demonstrated at all levels of modeling, starting from the behavioural level down to the gate level.
[1], paragraph 5.3.3 a. – c.

c) Architecture Evaluation and Optimization Report
Activities specified in the Architecture definition and to be verified according to the Verification plan, shall be covered by the Architecture Evaluation and Optimization Report.

d) Preliminary Data Sheet
A preliminary Data sheet shall be produced, updated and completed later at the end of the ASIC/FPGA development. (See chapter [1] ,7.a.1 for details)

e) Design Database, containing:
   - Simulation models
   - Verification results

f) Memory of meeting (MoM) of the Architectural design review (ADR)
The MoM shall in case of a satisfactory result of the ADR meeting provide the evidence that an authorization to proceed with the Detailed Design has been given. The MoM of the ADR shall be added to the Management Documentation.

7.1.3.3. Detailed Design

The high-level architectural design is translated during the Detailed Design phase into a structural description on the level of elementary cells of the selected technology/library.
The main output of the Detailed Design is a Design Database containing, or allowing an automatic and repeatable generation of the inputs to the Layout. The scripts required for this generation are an essential part of the Detailed Design, and all these scripts shall be part of the design Database.

a) Design Entry Report
This document shall cover, among others, the design tasks specified in the ADP, the implementation of the test concepts and the selection of buffers to the I/O requirements defined in the ASIC/FPGA Requirements Specification. See [1], paragraph 5.4.2 a – g.

b) Netlist Generation Report
This document shall cover the tasks carried out in order to translate the source description of the design into the netlist as well as any other information required for the layout generation.

c) Netlist Verification Report
Al the tasks required by the Verification Plan shall be covered. See [1], paragraph 5.4.4 a – l.

d) Updated Data Sheet with pin-out
The Data Sheet shall be updated to incorporate the new established information on pinout, estimated timing, etc. For further details see [1], paragraph 7.4.1

e) Updated Design Database, containing:
   - Pre-layout netlist
   - Constraints for layout (floorplan, constraints for timing driven layout, etc.) as defined in the ADP
   - Test vectors for production test

f) Memory of Meeting (MoM) of Preliminary Design Review (PDR)
The MoM of the PDR shall in case of satisfactory result of the PDR meeting provide the evidence that an authorization to proceed with the Layout has been given. The Preliminary Design Review shall as minimum cover and approve the design decision/trade-offs taken during the Detailed Design phase, the conformance to reliability, testability and radiation hardening requirements, and the extent and results from simulations in this phase. The risk mitigation activities shall be checked. The outputs to be reviewed and the items to be checked are listed in [1], paragraphs 1 – 5.

The MoM of this meeting shall be added to the Management Documentation.

7.1.3.4. Layout

The layout shall generate the placement and routing information to meet the design rules, timing and other constraints.

a) Layout generation Report
The Layout Generation Report shall cover the tasks carried out under the Layout generation phase. See [1], paragraph 5.5.2 a – I.

b) Layout Verification Report
The Layout Verification Report shall cover the tasks carried out under the Layout Verification phase. See [1], paragraph 5.5.3 a – l.

c) Design Validation Plan
The Design Validation Plan shall specify the measurements that shall be performed on the prototypes in order to verify that the implemented devices contain the functionality and the characteristics they are designed for. See [1], paragraph 5.5.4 a – d.

d) Updated Data sheet
The layout verification results in updated parameters that shall be documented in the Data Sheet. For further details see [1], chapter 7.4.1

e) Draft Detail Specification
The set-up of the Draft detail specification shall be based on the information collected in the Design documentation, and should comply to the requirements given in [1], chapter 7.4.3.

f) Updated design Database, containing:
Post-layout netlist in the agreed format depending on the targeted technological approach (GDS II, FPGA P&R files or other) Corresponding parasitic information

g) Memory of meeting (MoM) of the Critical Design Review (CDR)
The CDR shall result in the approval of design and layout and the release for prototype implementation. As a minimum it shall cover and approve the layout, final simulations and the production test, the Design Validation Plan and the risk mitigation activities. The outputs to be reviewed and the items to be checked are detailed in [1], paragraphs 1 – 6 of the current section.

7.1.3.5. Prototype implementation

Agreed number of tested devices (ASICs or FPGAs)

a) The number of prototypes to be produced for validation is specified.
b) Production test results and reports
c) Burn-in, any other production test results, specifications and patterns

7.1.3.6. Design validation and release

The Design validation and release shall generate the information concerning the results of the validation tasks and the final data sheet and Detail Specification. The documents associated with that phase shall contain:

a) Validation Report
b) Radiation Test Report (if applicable)
c) Release Report
d) Experience Summary Report
e) Final Data Sheet
f) Final Data Specification
g) Application Note
h) Memory of Meeting of Design Verification Review

This review shall result in the final acceptance of the design. As a minimum it shall cover and approve the production test and design validation test results and all documentation produced and updated in this phase.

- The completeness of the Design Validation Documentation together with the documentation of previous development phases is checked
- The device achievement of functional, performance, interface and compatibility characteristics satisfying ASIC/FPGA Requirements Specification is checked
- The existence of preventive measures and/or contingency plans for all identified risk items and the risk analysis of FM production is checked

The MoM of this meeting shall be added to the Management Documentation

i) Validation Breadboard
j) Burn-in or Screening Test Boards for FM parts

7.1.4. Management documentation

7.1.4.1. Objectives

The documentation shall provide the overall strategy for the development activity including task planning and organization, methods and procedures. In the present context, development of ASIC/FPGA is an integrated part of the system realization. What constitutes sufficient information will be dependent upon a number of factors, including the complexity and size of the E/E/PE safety-related systems and the requirements relating to the specific application.

7.1.4.2. General

The scope of this work is restricted to the activities related to the realization phase (block 9 of the safety lifecycle) of the HDL-based circuit. It is assumed that the preliminary analyses of the overall safety lifecycle have been carried out according to the methods and principles recommended by IEC 61508 (parts 1 – 7). The results of the hazard and risk analysis consequently establish the overall safety requirements and the safety requirements allocation.
The requirements listed in [2], clause 5.2, are compulsory in the development of HDL-based E/E/PE control systems at each applicable phase of construction.
The documentation shall contain sufficient information required for the management of functional safety.

7.1.4.3. Requirements

Following basic requirements are compulsory to documentation related to any parts of the system:

The documentation shall contain sufficient information required for the implementation of a functional safety assessment, together with the information and results derived from any functional safety assessment.

Unless justified in the functional safety planning or specified in the application sector standard, the information to be documented shall be as stated in the various clauses of IEC 61508.

The availability of documentation shall be sufficient for the duties to be performed in respect of IEC 61508.

The documentation shall:

- Be accurate and concise
- Be easy to understand by those persons having to make use of it
- Suit the purpose for which it is intended
- Be accessible and maintainable

The documentation or set of information shall have titles or names indicating the scope of the contents, and some form of index arrangement so as to allow ready access to the information required in IEC 61508.

The documentation structure may take account of company procedures and the working practices of specific application sectors.

The documents or set of information shall have a revision index (version numbers) to make it possible to identify different versions of the document.

The documents or set of information shall be so structured as to make it possible to search for relevant information. It shall be possible to identify the latest revision (version) of a document or set of information.

All relevant documents shall be revised, amended, reviewed, approved and be under the control of an appropriate document control scheme.

Where automatic or semi-automatic tools are used for the production of documentation, specific procedures may be necessary to ensure effective measures are in place for the management of versions or other control aspects of the documents.

7.1.5. Management of functional safety

Management and technical activities necessary to ensure that the E/E/PE safety-related system achieve and maintain the required functional safety shall consider the following issues according to [1], clause 6.2:

- The policy and strategy for achieving functional safety, together with the means for evaluating its achievement, and the means by which this is communicated within the organization to ensure a culture of safe working
- The identification of the persons, departments and organizations, which are responsible for carrying out and reviewing the applicable overall, E/E/PES or software safety lifecycle phases (including, where relevant, licensing authorities or safety regulatory bodies)
- The overall, E/E/PES or software safety lifecycle phases to be applied
- The way in which information is to be structured and the extent of the information to be documented (see previous paragraphs).
- The selected measures and techniques used to meet the requirements of a specified clause or subclause (see [3], [4] and [7])
- The functional safety assessment activities, see [2], clause 8.
- The procedures for ensuring prompt follow-up and satisfactory resolution of recommendations relating to E/E/PE safety-related systems arising from:
  - Hazard and risk analysis (see [2], clause7.4)
  - Functional safety assessment (see [2], clause 8)
  - Verification activities (see [2], clause 7.18)
  - Validation activities (see [2], clauses 7.8 and 7.14)
  - Configuration management (see [2], clauses 6.2.1 and 7.16, [3] and [4])

The procedures for ensuring that applicable parties involved in any of the overall, E/E/PES or software safety lifecycle activities are competent to carry out the activities for which they are accountable. In particular, the following should be specified:

- Training of staff in diagnosing and repairing faults and in system testing
- Training of operations staff
- Retraining of staff at periodic intervals

The procedures that ensure that hazardous incidents (or incidents with potential to create hazards) are analyzed and those recommendations made to minimize the probability of a repeat occurrence.

The procedures for analyzing operations and maintenance performance; in particular procedures for:

- Recognizing systematic faults which could jeopardize functional safety, including procedures used during routine maintenance, which detect recurring faults
- Assessing whether the demand rates and failure rates during operation and maintenance is in accordance with assumptions made during the design of the system

Requirements for periodic functional safety audits in accordance with this sub-clause, including:

- The frequency of the functional safety audits
- Consideration as to the level of independence required for those responsible for the audits
- The documentation and follow-up activities
- The procedures for initiating modifications to the safety-related systems, see [2], clause 7.16.2.2.
- The required approval procedure and authority for modifications.
- The procedures for maintaining accurate information on potential hazards and safety-related systems

The procedures for configuration management of the E/E/PE safety-related systems during the overall, E/E/PES and software safety lifecycle phases; in particular the following should be specified:

- Stage at which formal configuration control is to be implemented
- Procedures to be used for uniquely identifying all constituent parts of an item (Hardware and Software)
- Procedures for preventing unauthorized items from entering service

a) Where appropriate, the provision of training and information for the emergency services.
b) The activities specified as a result of a) shall be implemented and progress monitored.
c) The requirements developed as a result of a) shall be formally reviewed by the organizations concerned, and agreement reached.
d) All those specified as responsible for management of functional safety activities shall be informed of the responsibilities assigned to them.
e) Suppliers providing products or services to an organization having overall responsibility for one or more phases of the overall, E/E/PES or software safety lifecycles (see a)), shall deliver products or services as specified by that organization and shall have an appropriate quality management system.

7.2. Design process

This sub-section describes the design and verification of HDL-code, a major part of a safety-related control system. The basis of this text is the ASIC design process described in the draft standard, [1] currently used in the design process at DELTA electronics. In order to introduce safety aspects into the design process, relevant requirements from IEC 61508 (for ASIC design) have been retrieved and distributed among the steps in the design process. The required design flow from IEC 61508 (see Figure 6) is by this means mapped to the previously mentioned design flow.

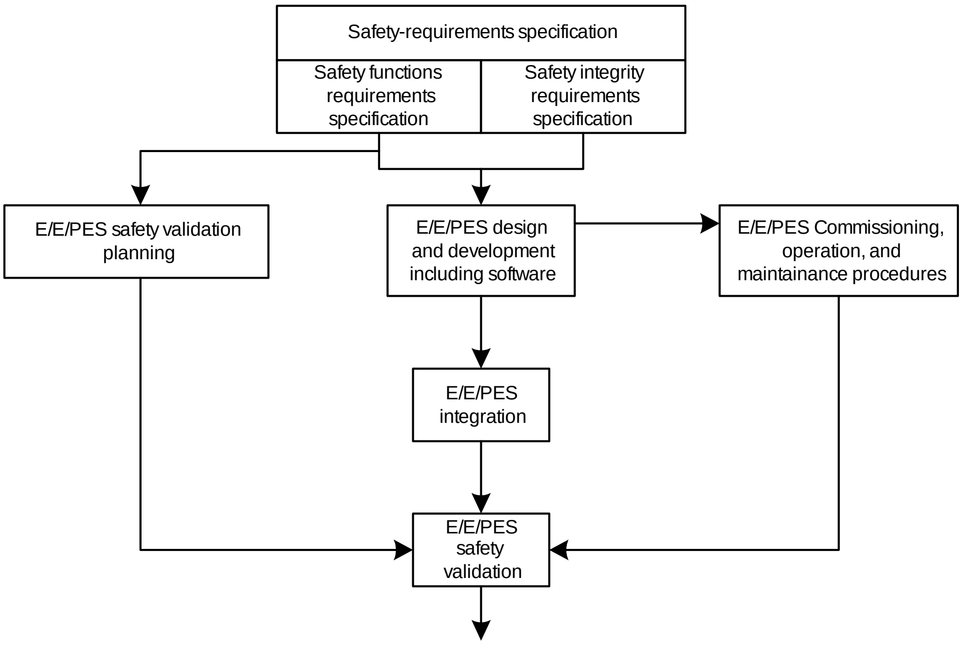

Safety-requirements specification
Safety functions requirements specification
Safety integrity requirements specification
E/E/PES safety validation planning
E/E/PES design and development including software
E/E/PES Commissioning, operation, and maintainance procedures
E/E/PES integration
E/E/PES safety validation

Figure 6: Realization phase in the safety life cycle (IEC 61508).

The design flow in Figure 6 is divided into following parts in this report:

a) Definition phase (Safety-requirements specification part in Figure 6)
   -Requirements on documentation
b) Architectural design (Design phase in Figure 6)
   -Requirements on the hardware design and development (process)
c) Detailed design (Design phase in Figure 6)
   -Requirements on the HDL specification
d) Layout (Design phase in Figure 6)
e) Implementation (Design phase in Figure 6)
   -Requirements on implementation and integration
f) Validation and release (Validation phase in Figure 6)
   -Requirements on safety validation

The design flow in [1] differs from the IEC 61508 realization flow (Figure 6) only in detail, and hence, is very suitable for mapping the standards together.

7.2.1. Definition Phase

The most important outputs from this phase are the following documents:
Related to the EUC (See [2])

a) Concept
b) Overall scope definition
c) Hazard and risk analysis
d) Safety requirements specification

Related to the control system (due to [1], clause 5.2)

e) Identification of ASIC requirements specification, including part of the safety requirements allocation specification
f) Feasibility study
g) Risk analysis (due to the quality assurance system)
h) ASIC development plan
i) Initial design review

All the tasks listed above have to be performed but are not considered in the method presented and described in this report. The associated documents are assumed to be available prior to making use of this guideline.

Regarding the safety-issues in this part of the design process, IEC 61508 emphasizes SIL's, a fundamental entity and primary criteria for safety validation. The SIL levels define the integrity of a safety function in its application (EUC) and may only be determined with a hazard and risk analysis. The SIL of an E/E/PES safety function depends on how much risk reduction is required for a specific application (EUC). If the same E/E/PES safety system is used for controlling different machines/applications (EUC), different SIL may be required from its safety function(s). The standard IEC 61508 divides safety functions into two types; low-demand and high demand safety functions. See Table 4 and Table 5 below. This also has an impact on the definition phase viewed in Figure 3. The overall ASIC design flow also has to be considered when developing the ASIC requirements specification.

Table 4: Safety integrity levels: Target failure measures (PFD) for a safety function operating in low demand mode of operation

Safety integrity Level	**Low demand mode of operation** (Average probability of failure to perform its design function on demand)
4	$\geq 10^{-5}$ to $< 10^{-4}$
3	$\geq 10^{-4}$ to $< 10^{-3}$
2	$\geq 10^{-3}$ to $< 10^{-2}$
1	$\geq 10^{-2}$ to $< 10^{-1}$

Table 5: Safety integrity levels: Target failure measures (PFH) for a safety function operating in high demand or continuous mode of operation

Safety integrity Level	**High demand or continuous mode of operation** (Probability of dangerous failure per hour)
4	$\geq 10^{-9}$ to $< 10^{-8}$
3	$> 10^{-8}$ to $< 10^{-7}$
2	$\geq 10^{-7}$ to $< 10^{-6}$
1	$\geq 10^{-6}$ to $< 10^{-5}$

The probability value ranges in Table 4 and Table 5 do not only concern hardware reliability, but the whole safety function, taking into account both hardware, software, the design process, operation and maintenance and many other factors.

The design of the safety-related PES shall be created in accordance with the E/E/PES safety requirements specification according to Figure 6. This safety requirements specification shall include tailored requirements on how to meet the requirements in the following subsections regarding:

- Hardware safety integrity
- Systematic safety integrity
- System behaviour on detection of a fault

7.2.1.1. General Hardware requirements

The requirements mentioned below are not limited to the ASIC design nor the subsystem design, but shall be considered during the specification relevant to both design processes for safety-related systems.

Following requirements for estimating the probability of random hardware failure for safety functions have to be regarded when designing the subsystem. The probability of failure of the safety functions shall be equal or less than the target failure measure as specified in the safety system requirement specification (ref. [3], clause 7.4.3.2.1). Although the requirements mentioned below are not limited to the ASIC design, they have to be included in the ASIC design process, since the ASIC takes responsibility for a large part of a safety function.

Requirements for the diagnostic test interval
a) The requirements for the diagnostic coverage are related to the control of the subsystem, and therefore involve the functionality of the ASIC and the subsystem configuration. The requirement for the diagnostic test interval of a subsystem is divided into the following three possibilities
Hardware fault tolerance > 0

1) Hardware fault tolerance = 0 (subsystem on which the safety function is entirely dependent) Used in Low demand mode
2) Hardware fault tolerance = 0 (subsystem on which the safety function is entirely dependent) Used in high/continuous mode

For 1) and 2) the DC test interval shall be selected in order to enable the E/E/PES to meet the requirement for the probability of random hardware failures (see Table 8)
For 3) The sum of the DC test interval and the fault reaction time shall be less than the process safety time where the process safety time means the time between failure occurrence (in the EUC or EUC control system) and the occurrence of a hazardous event if no safety function is performed.
Ref. [3], clause 7.4.3.2.3, 7.4.3.2.4 and 7.4.3.2.5

Requirements for system behaviour on detection of a fault
b) The detection of a dangerous fault in a subsystem that has:

1) Hardware fault tolerance > 0 shall result in either:
    - A specified action to achieve and maintain a safe state
    - Isolation of the faulty part of the subsystem to allow continued safe operation. If the faulty part is not repaired within the assumed MTTR, the system operation shall result in 1).
2) Hardware fault tolerance = 0 (low demand mode) shall result in either:
    - A specified action to achieve and maintain a safe state.
    - The repair of the faulty subsystem is performed within the assumed MTTR. During this time the safety of the EUC shall be ensured by additional measures and constraints that shall provide at least the same risk reduction as the E/E/PES safety-related system without the presence of faults. If the repair is not performed within MTTR, the system operation shall result in 3).
3) Hardware fault tolerance = 0 (high/continuous mode) shall result in
   a specified action to achieve and maintain a safe state

The detection of faults may be done by diagnostic tests, proof tests or by other means.
Ref. [3], clause 7.4.6

Requirements for the control of systematic failures
c) For controlling systematic faults the E/E/PES shall be designed to be tolerant against:

- Any residual design faults in the hardware
- Environmental stresses including electromagnetic disturbances
- Mistakes made by the operator of the EUC
- Any residual design faults in the HDL (software)
- Errors and other effects arising from any data communication process

d) The design of the safety-related E/E/PES shall regard the human capabilities and limitations. The design of all interfaces shall follow good human-factor practice.
The design shall accommodate the likely training or awareness of operators.
Ref. to [3] clause 7.4.5

e) The principles mentioned in section 6.2 about fault models, section 6.3 about redundancy and section 6.4 about diagnosis shall be considered during the architectural design due to above mentioned requirements.

General requirements on the design performance
f) Appropriate techniques and measures shall be used during the design and development of the hardware. Refer to [3], table B.2 This table contains recommendations for usage of e.g. structured design, modularization, checklists etc. in order to avoid faults during design and development. This shall be followed both for the subsystem design and the ASIC design.

g) The requirements in [3], clause 7.4.4.2 – 7.4.4.6 regards the set of documentation that shall be produced in order to prevent the introduction of faults during the design and development. These requirements regard facilities of the chosen design method, maintenance, and type of tools to be preferred and are as applicable for the subsystem design as for the ASIC design.
Strategies and means for fulfilling these requirements shall be added to both the ASIC design specification and the subsystem design specification in order to meet the requirements for the avoidance of failures.

h) Maintainability and testability shall be shall be considered during the design, according to the detailed system requirements specification.

i) De-rating shall be used as far as possible for all components. Where de-rating is appropriate, a de-rating factor of at least 0.67 should be used.

7.2.1.2. General ASIC design requirements

All the above requirements have to be reflected in the HDL/ASIC design. As the HDL design is a software model of later hardware, the safety requirements for the HDL design as such should apply as if the HDL model was in fact a hardware design. So the safety requirements for the HDL model are actually those for the hardware design.

Since it is a software model of hardware that is designed, only some requirements of [4] apply. In general, the requirements concerning the HDL development and verification process apply (with the restrictions imposed by the fact that HDL and common programming languages have different features and applications), whereas the general safety requirements and validation requirements do not. The reason for this is that the HDL design is developed similarly to conventional software, but is transformed in hardware before being operational for validation or system integration. However, if the HDL design contains firmware (software running on an embedded CPU), all requirements of [4] must apply to that part of the HDL design.

When designing an ASIC most of the transformations from the HDL model to the finished chip are automated using specialized software tools. These tools can be directed through control scripts, but because the number of design parameters and the data size are considerable, it is not feasible to control the tools in all details. Therefore it has to be envisaged to ensure complex safety critical functions at the architectural level rather than at the detailed level.

As the entire design exists electronically throughout the entire design work, first as HDL, later as databases and tool control scripts, it is required that all source files (HDL and scripts) that are needed for building the design are kept under version control. Keeping all source files under version control enables structured HDL development, keeps track of different versions and eases development logging for documentation purposes.

Designing safety critical systems thus relies on the safety functions being sufficiently specified before commencing the HDL design since the real design is performed through the HDL modeling. Careful surveillance of the subsequent design transformation steps is required, so that the intended safety functions are implemented correctly. This can be done through constraining the tools or by manual interaction with the tools; this should be reduced to a minimum by choosing the right architecture from the beginning.

All the redundancy principles from section 6.3 can be implemented in HDL. Software redundancy can be seen as algorithms or control flows which basically can be translated to HDL. This means that clever software safety algorithms could be used in the HDL design. Another way of making the design robust is to add fault tolerance or fault recovery. One example is to add error correction codes to e.g. memories, which enable correction of bit errors on the expense of a few extra data bits.

It is important that the ASIC is not accidentally set in an illegal state as a consequence of unstable or noisy signals. The behaviour and noise (including glitches) of external input signals must be evaluated. Noisy input must be filtered and all asynchronous signals must be synchronized to the design’s internal clock before being used.

In order to ensure that the ASICs are manufactured correctly, it is required that the ASICs are tested after manufacture with a high coverage of possible faults. To achieve high fault coverage, it is required that the design is prepared for the manufacture test already during the design phase, a step called *Design For Test*. This step has nothing to do with functional verification during design, but enables testing that the ASIC has been manufactured without errors. Fault models are used to express the possible manufacturing errors. Most commonly used is the *Stuck-at* fault model, which models an error as either the input or output pins on a gate being tied to power or ground (logically 1 or 0). Usually an overall fault coverage of more than 95% is acceptable, but in case of safety critical designs, the fault coverage should be much higher. This calls for careful design of safety functions as well as special test preparations of the design to enable high fault coverage of these parts of the design.

When it comes to the technology used for the ASIC, it must be evaluated with regard to robustness towards the environment it shall operate; high voltage applications need high voltage technologies, space applications need special space approved technologies. Furthermore, a well known and stable design library for the chosen technology should be used for implementing the ASIC, in order to avoid failures due to lack of validation and characterization of the library cells.

The result of this phase gives a complete set of technical documentation, including a description of the EUC and the interfaces between the EUC and the safety-related control system. In addition to this a risk analysis has been performed providing a specific SIL for each safety function and a safety-requirements specification and allocation. The required documentation from the [1] draft standard provides with in detail studies on general design requirements, feasibility, a development plan and an initial design review that prepares for the next phase – architectural design.

7.2.2. Architectural Design

Before engaging the design process measures have to be applied for avoiding systematic faults. A systematic fault is a fault introduced in the design flow that propagates through the design process in the manufactured end-product. Examples of such failures are if a designer by mistake renames a signal erroneously to another signal or if a manufacturing machine always connects a bonding wire to an erroneous pad. These kinds of failures become systematic first when the product reaches the end-user. It is possible to reduce the probability of systematic failures by careful testing/verification and by using special design features (see A.3 in [3]). Following method for selecting a design approach for hardware and systematic safety integrity is noted in [3].

a) Determining the SIL of the safety functions (Refer to [2]), should been done during the definition phase.
b) Hardware safety integrity = Systematic safety integrity = SIL (Table 4 and Table 5)
c) Hardware safety integrity: Determine the architecture and calculate the probability of loss of the safety function due to random hardware single faults (see below)
d) Systematic safety integrity:
    1) Use design features that tolerate systematic faults
    2) Confirm fulfillment of requirements on “proven-in-use” components
    3) Use techniques and measures for preventing introduction of systematic faults

From the Requirements Specification developed during the Definition Phase, the actual design implementation work is initiated. The design flow is depicted in Figure 7 which shall be followed during the design and implementation. Usually the work during this phase follows two paths: Establishing the ASIC design and establishing the test benches in parallel. The reason is that it is not uncommon that much more time is spend on test benches than on the actual ASIC design work itself, so it is important to initiate the test bench verification early in the phase.

Most commonly used HDL design development strategy is a top down strategy. Based on the architectural investigations done during the *Definition Phase*, the basic structural hierarchy of the design is established. This *Architectural Definition* is the foundation for the rest of the design and considerations about the subsequent physical implementation of functions and blocks has to be taken into account here. There are means for re-organizing the design hierarchy during synthesis, but it is far better to create the right architecture from the beginning.

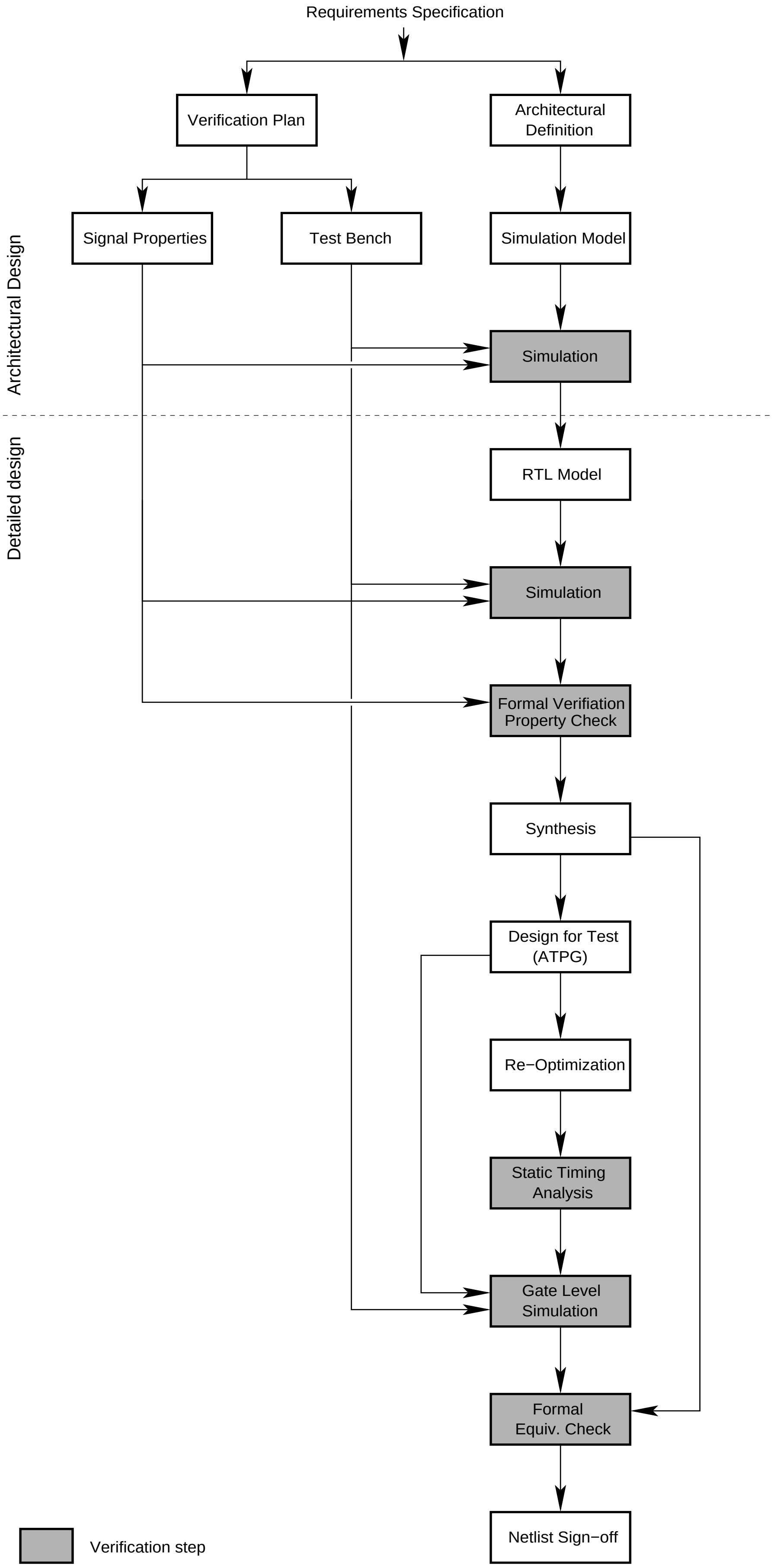


Figure 7: Architectural and Detailed Design Flow.

7.2.2.1. Requirements on the architectural design

The design shall be based on a decomposition of subsystems/modules with each subsystem/module having a specified design and a set of integration tests. This also includes the design of the target ASIC. If a subsystem has multiple outputs and if a specific combination of the subsystem output states may cause a hazardous event, the prevention of this output state(s) shall be regarded as a safety function operating in high/continuous mode of operation.

The architectural definition does not only characterize the system in relationships between sub-modules and fundamental functions but does also define fault tolerance of the complete subsystem. According to IEC 61508 there is no support for on-chip fault tolerance so the ASIC has to be designed taking into account redundant modules outside the ASIC according to Figure 8 since a large part of the safety function is to be implemented in the ASIC.

Table 6 and Table 7 specify the highest safety integrity level that can be claimed for a subsystem (see Figure 8) carrying a safety function. Taking into account the level of fault tolerance and the safe failure fraction, the requirements from these tables shall be applied on every subsystem carrying a safety function and hence every part of the E/E/PE safety-related system. So although the ASIC is one out of several redundant channels the architectural constraints on the subsystem as such impacts on the ASIC internal architectural design. The more the safety function depends on the ASIC internal functionality the more these architectural requirements affect the ASIC architectural design.

- Hardware fault tolerance N means that N+1 faults may cause the loss of the safety function
- If one fault leads to the occurrence of one or more subsequent faults these are considered as a single fault
- When determining the hardware fault tolerance some faults may be excluded if the probability of their occurrence is very low. A fault exclusion shall be properly documented and justified
- The safe failure fraction (SFF) is defined as the ratio of safe failures + dangerous detected failures to the total average failure rate of the system.

The tables below regard subsystems as shown in Figure 8. The most important parameter to consider during the ASIC architectural design phase is the level of hardware fault tolerance.

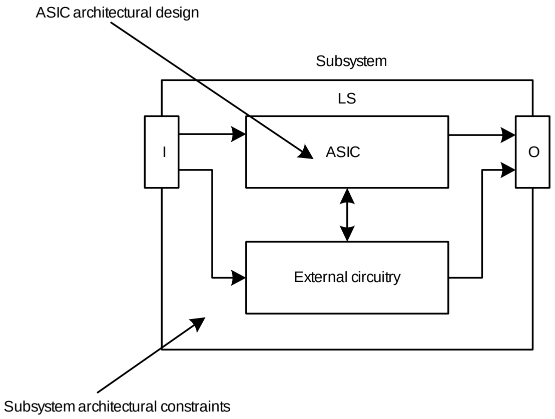

ASIC architectural design
Subsystem
LS
I
ASIC
O
External circuitry
Subsystem architectural constraints

Figure 8: Subsystem architectural constraint and ASIC architectural design.

a) Type A safety-related subsystems –

1) The failure modes for all components are well defined
2) The behaviour of the subsystem under fault conditions is well determined

3) There is sufficiently dependable data from experience to claim the rates of detected and undetected dangerous failures

This table is not applicable for such subsystems (type B) that falls in the scope of this report but are useful is such a subsystem are used in parallel with a type A subsystem and is hence not excluded from this report.

Table 6: Architectural constraints on type A subsystems

Safe Failure Fraction	**Hardware Fault Tolerance**		
	0	1	2
< 60%	SIL1	SIL2	SIL3
60% - < 90%	SIL2	SIL3	SIL4
90% - < 99%	SIL3	SIL4	SIL4
≥99%	SIL3	SIL4	SIL4

b) Type B safety-related subsystem –

1) At least one component in the subsystem has not well defined failure modes
2) The behaviour of the system under faulty conditions cannot be completely determined
3) There is insufficiently dependable data from experience to claim the rates of detected and undetected dangerous failures

This table shall be used in order to determine the architecture of the subsystem in which the ASIC is embedded.

Table 7: Architectural constraints on type B subsystems

Safe Failure Fraction	**Hardware Fault Tolerance**		
	0	1	2
< 60%	N/a	SIL1	SIL2
60% - < 90%	SIL1	SIL2	SIL3
90% - < 99%	SIL2	SIL3	SIL4
≥99%	SIL3	SIL4	SIL4

Both Table 6 and Table 7 are applicable for E/E/PE safety-related systems comprising both type A and type B subsystems. In a E/E/PE safety-related system where the safety function are implemented by a single channel of subsystems, the hardware safety integrity level that can be claimed is determined by the subsystem with the lowest hardware safety integrity level

In an E/E/PE safety-related system where the safety function is implemented through multiple channels of subsystems the resulting hardware safety integrity level shall be determined by:

1) Assessing each subsystem against Table 6 and Table 7
2) Grouping the subsystems into combinations
3) Analyzing those combinations to determine the overall hardware safety integrity

7.2.2.2. Requirements on the probability of hardware failures

Following parameters shall be taken into account when estimating the probability of failure for each safety function
Ref. [3], clause 7.4.3.2.2

Table 8: Fundamental hardware requirements

Ref.	Parameter	Comment
a)	E/E/PE safety-related system architecture as its relates to each safety function	Deciding which failure modes are in a series configuration and which are in a parallel configuration
b)	λ_{DD}	The estimated rate of failure that would cause a dangerous failure to the E/E/PES but which are detected by diagnostic tests
c)	λ_D	The estimated rate of failure that would case a dangerous failure to the E/E/PES which are undetected by the diagnostic tests
d)	Susceptibility of common cause failures (β)	A common cause failure is such a failure that causes coincident failures in two or more channels in a multiple channel systems. Refer to [6], annex D for guidelines for quantification of common cause failures
e)	Diagnostic coverage (DC) and the diagnostic test interval	The DC is calculated by DC = $\sum \lambda_{DD} / \lambda_D$ The diagnostic test interval and the subsequent time for repair constitutes the mean time for restoration. Also refer to [3] clause 7.4.3.2.2 e) and annex C).
f)	Interval of proof test	The interval at which proof tests are undertaken to reveal faults that are undetected by diagnostic tests.
g)	Repair times for detected failures (MTTR)	See e) and [3], clause 7.4.3.2.2 g)
h)	λ_D for communication	Refer to [3], clause 7.4.3.2.2 h)

7.2.2.3. Requirements on verification and validation planning and initiation

First step in the verification and validation is establishing the *Verification and validation plan* that contains a description of how the ASIC design and its corresponding subsystem is supposed to be verified and in the end validated, this corresponds to the safety validation planning in Figure 6.

The subsystem verification planning shall consider:

- The selection of verification strategies and techniques
- The selection and utilization of test equipment
- The selection and documentation of verification techniques
- The evaluation of verification results

The safety validation planning shall be carried out concurrently with the system architectural and detailed design and is required to include the following:

Before engaging the safety validation a plan shall be established in order to specify those steps the safety validation shall be divided into. This plan shall consider the following:

a) all requirements in the E/E/PES requirements specification
b) The procedures to be applied to validate the implementation each safety function with their corresponding pass/fail criteria

c) The procedures to be applied when validate the safety integrity of each safety function with their corresponding pass/fail criteria
d) The environment in which the tests are to be carried out and testing equipment
e) Test evaluation procedures
f) Test procedures and performance criteria for validating the specified EMI criteria
g) Policies for resolving validation failures

There are several ways to verify an HDL design: simulation, emulation, prototyping in FPGA or formal methods. The *Verification Plan* shall also define how to verify the ASIC in the subsequent design phases, in order to keep the functionality of the transformed ASIC design consistent with both the RTL code and the requirements specification. This verification plan shall also include the E/E/PES verification plan. That verification plan includes the criteria, techniques and tools to be utilized when passing through the design phases due to the requirements of safety integrity (Ref. [3], clause 7.9.2.1 – 7.9.2.4). The E/E/PES verification plan becomes a natural but also required part of the ASIC verification plan.

Simulation is the most widely used way of verifying the ASIC design. It has the advantage that the design can be observed in depth, and that simulation can be launched on sub designs early in the design phase. The creation of simulation test benches are crucial for the verification of the design in the subsequent phases, where the HDL code is transformed to hardware. The disadvantage of simulation is the sometime long simulation runtimes; this can in some cases be reduced by using emulators, which is a mixture of FPGA prototyping and simulation, where hardware emulates the design under simulator control. However, the hardware emulators are often very expensive.

As the need for verification of the ever growing ASIC design complexity, new methods for verification based on formal methods have emerged. The formal methods aim at proving through math that the HDL code functionality is equal to the specification. This methodology is still quite immature, lacks sufficient tool support, and relies on the ability to specify the behaviour in a high level language other than the HDL. Best established currently is the *Signal Property Checking*, where the focus is to describe the behaviour of signals, usually the most important ones, and check their behaviour during simulation or by carrying out a formal proof of conformance to a *Signal Property Specification* (written in *PSL* language), also see [18].

The ASIC verification planning is covered in more details in the following sections.

7.2.2.4. Requirements on the architectural HDL modeling and test bench design

When the design architecture is established, the HDL coding can begin. This can include direct RTL coding, behavioural modeling or incorporation of third-party designs (e.g. IP's, Intellectual Property). If needed, portions of the design can be implemented using a high level HDL description that subsequently can be refined to an RTL description. The goal of the *Simulation Model* step is to have a simulation model of the whole design. There is no requirement as to which level of detail sub modules of the design should be modeled, some modules can be in RTL and some as high level behavioural description. However, all critical modules should be modeled sufficiently to define their exact implementation. All hardware shall be treated as safety-related unless independency between safety functions and non-safety functions can be shown. If independence is required between safety functions this shall be documented and justified.
Ref. [3], clause 7.4.2.5

Throughout the coding phase of the design it is important to keep a good coding practice. This includes writing HDL code that is easy to understand, verify and maintain. As for conventional software it is important to apply a good coding standards a set of rules for how to write good quality code) and to document the code through comprehensive formatting, meaningful and unambiguous signal, variable, function and module naming. Clear and intuitive code shall be preferred to incomprehensible compact coding. The keyword is to keep the HDL simple and clean! Choosing a good hierarchical decomposition, proper signal and variable types, clever data structures and as high a level of abstraction as possible (not sacrificing the detailed safety design requirements) will help safe HDL modifications subsequently. Throughout the code development documentation shall be maintained in form of well commented HDL code and as a part of the design documentation.

These requirements apply for high level modeling, HDL test bench development and low level RTL modeling,- both levels shall be maintainable at all stages of the design flow. As most of the HDL transformation stages are controlled by command scripts (software), it is equally important that rules for good coding practice are applied here. The HDL transformation tools use different script languages; they can be standard languages as TCL or proprietary. Usually these scripts consist in sequential execution of different transformation tasks within the tool scope, and the level of programming is usually quite concrete and simple (no data structures etc.). The transformation tools are chosen and documented as a part of the ASIC development plan to cover the design process in the optimal way; this also implicitly defines the suite of programming languages used for the ASIC development.

To ensure a consistent and transparent design development some form of version control must be applied. This ensures that all modifications have a history, are traceable and can be undone/redone if necessary.

During the design phases the HDL design shall be reviewed, both in terms verifying the correct implementation of the functionality but also looking at the code implementation with regards to the chosen coding standards. This applies also for the scripts used for the HDL transformations.

The result of this phase shall be an architectural design of the internal functionality of the ASIC with the subsystem architectural constraints taken into consideration.
The output of this phase shall include:

- E/E/PES architectural design
- Subsystem architectural design due to the mentioned constraints (e.g. tape A or type B subsystem)
- ASIC Internal architectural design and descriptions
- All parameters in Table 8 (for the whole subsystem)
- A simulation model of the ASIC design
- Verification plan (all internal sub-modules in the ASIC)
- Integration plan (How to integrate all sub-hardware modules, internally in the ASIC and in the subsystem)
- Safety validation plan (Initiated, considering the subsystem and the ASIC architectural design)
- Required documents

7.2.3. Detailed Design

In this phase the subsystem and the ASIC is detailed designed according to the architectural design specifications. During this phase the hardware safety requirements has to be considered. These requirements impact primarily on the subsystem but also on the detailed design of the ASIC. Figure 7 depicts the steps to carry out during the *Detailed Design* phase of the ASIC design. The input from the previous design phase is a simulation model of the entire design. This model shall have been verified to be consistent with the requirements specification from the definition phase.

The goal of the *RTL Model* step is to get a synthesizable HDL model. Modules of the design which have been modeled using high level behavioural constructs should be redesigned to RTL that can be synthesized. This requires the designer to be familiar with the language restrictions in the synthesis tool (not all HDL constructs can be translated to hardware), so the design work can include trial synthesis iterations for verification. This transformation is followed by verification by simulation, ensuring that the new model still behaves as required.

Prototyping in FPGA is another way of design verification and includes a full synthesis of the design targeted for the FPGA technology. The advantage is that the ASIC design functionality can be tested “real time” in the final environment (subsystem), thus avoiding very long simulation times. However, the ASIC design needs to be quite complete before the FPGA prototyping is recommended, because the means of design debugging is quite limited and cumbersome, when all signals are hidden away inside the FPGA.

First step is the translation of the RTL model to logic hardware primitives (*gates*) found in the target process standard cell library. The technology process and silicon foundry has already been decided in the *Definition Phase*, and the technology setup for the hardware implementation of the design is a prerequisite. Translating the RTL model to a physical representation of the design is called *Synthesis.* Synthesis involves transforming the RTL model into a mathematical representation which can be optimized.

The mathematical representation is then replaced by an identical representation, but this time consisting of hardware primitives from the technology library; this is called technology mapping. Upon technology mapping, the design is optimized, which involves choosing not only one implementation of the design, but choosing the optimum implementation between all available gates in the technology library. The optimization uses mostly two types of constraints: area and timing. The most important is to get the timing right, mostly dictated by the specified clock frequency. When the timing is fixed, then the area is minimized, while keeping the timing correct. Throughout the optimization, the drive capacity of all gates is observed, such that no gate must drive more gates than it is supposed to; otherwise the gate is replaced by one with higher driving capacity. It is important that the optimization algorithms keep any implemented redundancy between sub-modules so the detailed design remains to correspond with the architectural design. Modifications in the HDL-code might be necessary in order to achieve this. The output from this step is a *gate level netlist*, consisting of gates only (hardware primitives), stitched together by wire connections to form a physical representation of the design.

After synthesis, or sometimes during, post production test support logic is inserted, called *Design For Test*. Usually this consists of hooking all flip-flops in the design up into large shift registers forming so-called scan chains. Scan chains gives in-depth access to the design, such that all gates in the design can be tested functionally after production. The design for test is handled by a tool that is able to automatically manipulate the gate netlist in order to get high overall testability of the design. Along with the design for test step, a test program using the scan chains is developed. This is done by another tool that analyzes all gates in the design and how to access them, how to test them and generates a test vectors that covers a maximum all possible manufacturing faults in the design. High detection coverage of possible manufacture faults is required in order to ensure that the chip has been manufactured correctly before the chip is mounted in the application.

Both the synthesis and the subsequent design for test interacts closely with the RTL model, since if either fails, it is usually best to go back to the RTL model and do modifications. This usually means that several iterations from RTL coding to design for test are done during the design development.

Since the design for test manipulates gate netlist by inserting test support logic, it is necessary to *re-optimize* the design, ensuring that the additional test logic has not ruined the integrity of the design. The synthesis is concluded by a *Static Timing Analysis* that verifies that all the timing requirements in the design are fulfilled both for best case and worst case operating conditions. After this step the design is verified timing wise, leaving the functionality to be verified.

For verifying that the design’s functionality has been maintained throughout the synthesis either simulation or *Formal Equivalence Check* can be used. The simulation is performed using the test benches used during the Architectural Design phase and applies “real-life” stimuli to the design. Simulation is done on the gate level netlist, where gates are replaced with simulation models, and where gate and wire delay estimates are applied through a timing file (SDF, Standard Delay Format). This verifies that the design behaves correctly with the test benches both functionally and timing wise. However, since it is not uncommon to experience that simulations slows down 15 times when doing gate level simulations, it can be necessary to choose carefully between the functional test benches to re-run on the gate level netlist. One mandatory test bench is the post production test program, since it is very dependent on the test logic introduced during design for test, and also verifies that this test logic works as expected.

Formal Equivalence Check is another way of verifying that the synthesis transformation of the design has maintained the intended functionality. This is done by comparing the RTL model and the final gate level netlist for equivalence. This method proves that all the logic constructs in the RTL model have been correctly translated into gates, but does not consider the timing of the design. Combined with a *Static Timing Analysis*, the formal equivalence check could replace most of the gate level simulation. However, most commonly, simulation is still performed, since it more easily enables debugging and gives a more comprehensible display of correct design behaviour.

The integration testing applies both to the integration of the sub-modules in the ASIC and the integration of modules in the subsystem (including the ASIC when finished). In order to process the verification of the sub-modules and the system integration testing the modified V-model is useful and should be followed (see Figure 11).
Documentation shall be provided for each step of the modified V-model and the integration tests shall be supported by available information about how the hardware safety integrity requirements are met. In addition to this following procedures are required by IEC 61508 and shall be considered whenever applicable:

a) The E/E/PE safety-related systems shall be integrated according to the specified E/E/PES design and shall be tested according to the specified E/E/PES integration tests that were defined during the architectural design phase.
b) All modules in the E/E/PE safety-related system shall be tested and shown to interact correctly in order to perform its intended function.
c) Software shall be integrated according to [4]
d) If a failure is detected, the reason for the failure and its correction shall be documented
e) If a module is modified, an impact analysis shall be performed and possible re-verification
f) Required integration testing documentation contain following information
    1) version of the used test specification
    2) criteria for acceptance of integration tests
    3) E/E/PES or subsystem version
    4) Tools/equipment and calibration data
    5) The result of each test
    6) Any discrepancy between expected and actual results
    7) The analysis made and decisions on continuing tests or modify the test object
g) Techniques in table B.3 in [3] shall be used in order to avoid failures when performing the integration.

When the design (subsystem design and ASIC design) has been sufficiently verified and integrated, the layout design is initiated. The subsystem PCB layout shall be designed according to the detailed design specification and the result of the verification tests and integration tests.

Before the ASIC design layout can begin, a layout specification has to be made, additional formatting of the netlist is carried out and all the design constraints for the layout should be defined. This is all a part of the task signing off the design before layout ensuring that cumbersome and time demanding iterations can be avoided.

7.2.4. ASIC Layout design

The layout is the final phase in the design development. This is where the gate netlist is translated to a physical representation of transistors, wires, diodes and other layout primitives. The process flow that shall be followed is shown in Figure 9.

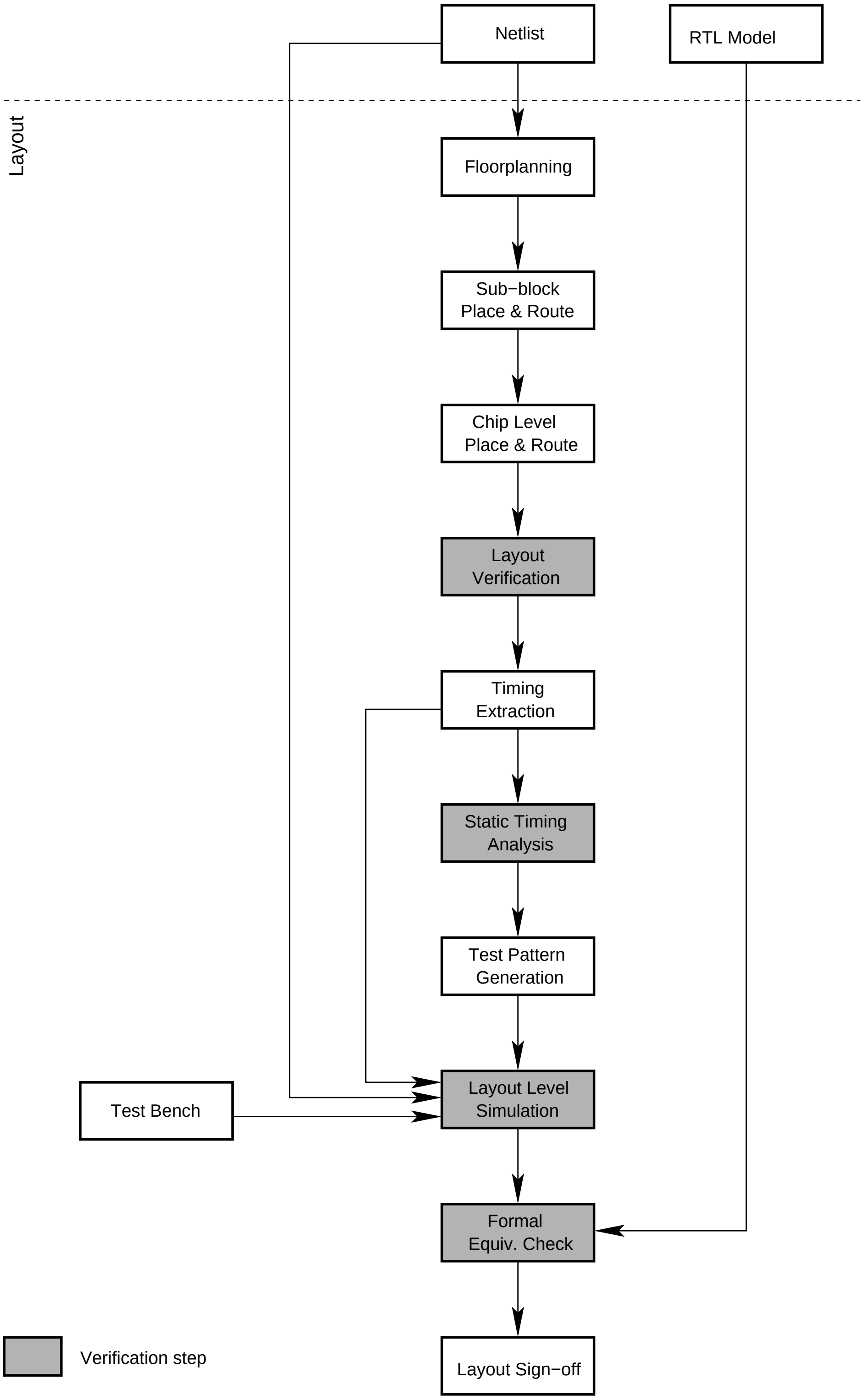


Figure 9: Layout Design Flow.

First step in the layout is to plan how to organize the design on the chip, what is called *Floorplanning*. Memory blocks and other large sub designs shall be placed in the best way to minimize the layout area and to facilitate the wire connections between the individual blocks. Sometimes it is desirable to subdivide the whole design into functional blocks; area for these blocks that need to be implemented as separate layout blocks is allocated in the floorplan. Different power supply areas should also be placed as separate areas on the chip. The chip pin-out is also implemented during floorplanning. Not all of the floorplanning is decided here, much of the work has already been done during the previous design phases, such that the gate level netlist already, to some extend, reflects the planned chip layout.

When the block layout of the chip has been decided, the standard cell layout can be generated. A standard cell layout consists of the physical implementation of the netlist (or parts of it), where the gates are replaced by their physical implementation and wire connections are replaced by metal wires in several layers (similar to a PCB). The physical gate consists of a configuration of transistors realizing the intended functionality and implemented in a way that enables gates to be placed in rows, one beside the other. All standard cells are placed and routed according the specified constraints, such that the size and shape of the block is correct and that the timing requirements are fulfilled. Insertion of buffers for correct timing of high fan-out nets such as clock and reset nets is also handled here.

Sub-block Place & Route is the implementation of standard cell layouts for sub blocks of the design. This division can be useful for better control over the layout process and timing, or if the same block has to be used more than once or for reducing the layout complexity of the entire chip by subdivision. These sub blocks are then assembled with the rest of the blocks on the chip (e.g. memories etc.) according to the floorplan in the succeeding *Chip Level Place & Route*. Here, all wire connections are routed between the blocks and the cells driving the chip pins (pad cells) are added. After this step the chip layout is finished and the rest of the tasks are verification steps.

The *Layout Verification* consists usually of two checks: *DRC* (Design Rule Check) and *LVS* (Layout Versus Schematic). DRC checks that the layout has been made in a way that can actually be manufactured, i.e. that the set of layout design rules for the process has not been violated; an example could be the minimum allowed distance between two metal wires. LVS is an equivalence check between the gate level netlist and the layout. This checks that the layout implements the gate netlist and that there are no short circuits between wires in the design. Passing these two layout checks are mandatory to all layouts.

To prepare simulation and static timing analysis of the layout, actual timing delay is extracted from the layout. When simulating the gate netlist after synthesis, timing delays have been estimates based on assumed wire lengths. Now actual wire delays can be extracted and used for precise delay calculations for *Static Timing Analysis* and *Simulation*. *Timing Extraction* is usually done by the layout tool along with the layout netlist (containing added net buffers etc.).

Because the layout tool may have added additional buffers or have re-arranged small portions of the logic, the final scan test program is not generated until now. The result *Test Pattern Generation*, also called *ATPG* (Automatic Test Pattern Generation) is simulated along with a selection of the test benches developed in the *Architectural Design* phase to verify that the functionality has been maintained through the layout phase. As after synthesis, the layout netlist can then be equivalence checked against the RTL model to verify that all logic functions have been translated to the physical implementation.

Last step in the layout phase is to sign-off the layout to the foundry and to prepare the encapsulation in the package required by the specification by defining how the chip is to be placed and connected in the package. This is the last chance to make any last minute changes to the layout before sending the design to manufacturing. Test programs for the post production test are also finalized and verified through simulation and a test specification defining operating conditions etc. during test is written.

7.2.5. Implementation

7.2.5.1. ASIC implementation

The steps in the physical prototype implementation will only be covered briefly. The layout is represented as a database containing all the layer definitions that is needed to implement the chip in silicon. As mentioned earlier, the manufacture is similar to photo lithographical printing, so the first step is to produce the “negatives” defining the layers, called masks. The number of masks depends on the foundry process and may vary from around 10 to more than 30. In modern processes, the mask set is quite expensive, so foundries have programs where multiple customers share cost of the mask set for prototype production.

After mask set production masks are “printed” on to silicon discs, the so called silicon wafers. The wafer is “stamped” with prints of the chip layers, one beside the other. This is done up to several times for each layer defined for the chip using very advanced technology. A wafer can contain several thousand chips in rows and columns, the so called chip dies.

When the wafer is produced, all the chip dies are cut from the wafer by slicing the wafer in rows and columns. For *encapsulation* each of the chip dies are picked from the sliced wafer and placed in the package *lead frame*, to which the inputs and outputs of the chip die are connected through thin gold wires (called bonding). After bonding the encapsulation is finalized by adding plastic mould to the lead frame thus forming the exterior of the package.

The final step in the manufacture of the chip is to test that the chip has been manufactured correctly. This is done by specialized test equipment where the test programs developed during the previous phase are used to assign stimulus to the inputs and to check the response on the outputs. These tests can be specified to be run at high or low temperatures or using various supply or input voltages. If the chip passes these tests, they are ready for shipment to the end user for *Design Validation and verification in Application*.

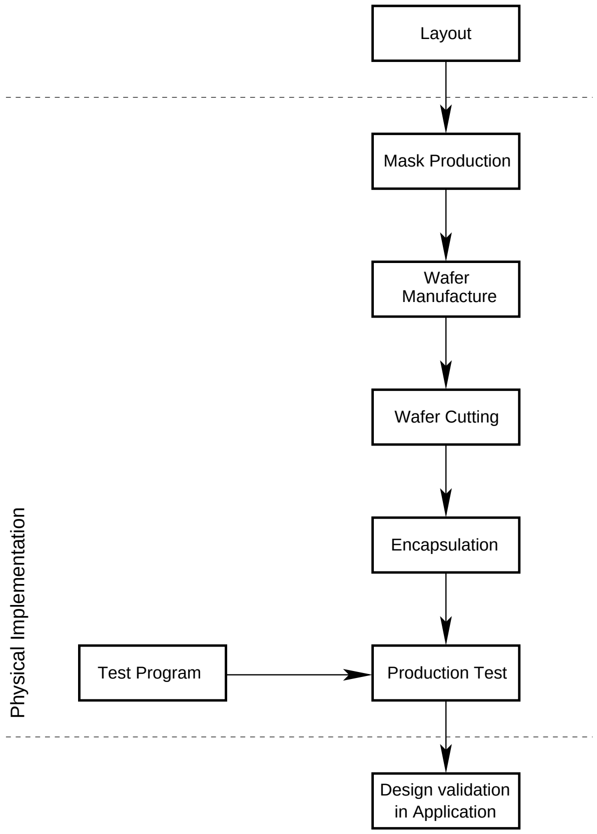


Figure 10: Implementation Flow.

7.2.5.2. System implementation

The implementation of the safety-related subsystem should be carried out in accordance with the E/E/PES design specification. All subsystems carrying safety functions shall be identified and documented.

a) Following documentation shall be available for each safety-related subsystems,

1) A functional specification of functions and interfaces that may be used as safety functions
2) The estimated rate of failure (random hardware failure) that would cause a dangerous situation and that is detected by diagnostic tests (λ_{DD})
3) Same as b) but are not detected by diagnostic tests (λ_{DU})
4) Environmental limits for the subsystem in order to maintain the validity of the estimated failure rates
5) Limit of the system life time in order to maintain the validity of the estimated failure rates
6) Any periodic proof tests and/or maintenance requirements
7) Diagnostic coverage (DC)
8) The diagnostic test interval
9) Information required for deriving a measure on the mean time-to-repair (MTTR), such as repair times
10) Information required for deriving the safe failure fraction (SFF) (see 7)
11) The hardware fault tolerance
12) Limits on the application subsystem in order to avoid systematic failures
13) The highest safety integrity level that can be claimed for a safety function which uses the subsystem on the basis of:
    - Measures ant techniques to prevent systematic faults to be introduced during the design and implementation of the subsystem
    - The design features which makes the subsystem tolerant against systematic faults

14) Information required in order to enable the configuration management of the E/E/PES
15) Documentary evidence that the subsystem has been validated

b) The estimated rate of hardware random failures may be determined by either

1) FMEA using component failure data from an industrial source (these data shall have a confidence level of > 70%). Site-specific data is preferred. The useful lifetime of the components shall also be noted (experience shows that useful lifetime often lies in the range 8 to 12 years).
2) From experience of previous use of the subsystem in a similar environment

c) Proven in use: a subsystem may be considered as proven in use if following conditions are fulfilled: (if this requirement is fulfilled no information about measures and techniques for prevention and control of systematic faults is required)

1) Clearly restricted functionality
2) Documentary evidence of previous use (where all failures has been documented)
3) The documented evidence shall demonstrate that the probability of failure (random hardware failure and systematic failure) is sufficiently low and corresponds to the specified SILs

Refer to [3], clause 7.4.7.7 – 7.4.7.12 for further requirements on proven-in-use subsystems.

8. System Verification and safety validation

The aim of the overall safety-validation is to prove that a function incorporated in a control system fulfils the specified safety requirements. In practice this means that the designer/assessor shall demonstrate that the control system has such an integrity that the end-user may trust the resulting function to be sufficiently safe and robust for its intended use.

In the context of the present field of application, the following characteristics of a control system are addressed:

- Hardware safety integrity
- Systematic safety integrity
- HDL safety integrity

These three properties are strictly dependent on the system design process (including documentation) and the resulting functionality of the control system (architectural design and detailed design). This may be defined as the functional safety by e.g. the two following specifications:

- The system functional properties specification (proven by validation)
- The system detailed design specification (proven by verification and test)

The specifications listed above are used only to enlighten some aspects concerning the validation process. In a realistic design both specifications are included in the system design specification.

The aim of the test and verification is to prove that the system conforms to the detailed design specification, i.e. that the current implementation corresponds to the intended design. Several aspects are considered during the verification such as the correctness of the synthesis result, the layout correspondence with the HDL description or the correctness of rule-checks. During the verification the functional correspondence with the technical specifications is also considered.

The aim of the validation is to prove that the application conforms to the specification of the functional properties of the system, i.e. that the function of the system is correct and sufficient according to the specified requirements. Such functional properties may be the system behaviour at fault, the conditions for reaching a safe-state or the level of fault tolerance. The validation process also considers if implemented measures for fault monitoring/control are sufficient for their intended purpose. An example of this is to validate an implemented volatile memory test algorithm. A validation is commonly a mixture of theoretical analysis techniques and functional tests in order to assure that a specific function is "valid" for its intended use.

The above mentioned approaches are included and required in the safety validation process. In order to make a reasonable judgment on the hardware-, the systematic- and the HDL safety integrity level one have both to assure that the design functions as intended and that the implemented functions are sufficient due to the EUC environment.

8.1. Requirements on system verification

In each design phase it shall be shown that the functional and safety integrity requirements are met. Methods for verification of the functional safety during the design are described in section 8.7.
The subsystem design verification shall include all design phases.
Ref. [3], clause 7.9.2.6 – 7.9.2.10 and table B.5 and table A.1

Working with HDL for defining the electronic circuit behaviour has a number of advantages over the traditional PCB design development. First of all the circuit behaviour can be fully customized realizing only the needed functionality. Secondly the functionality can be simulated and debugged before being implemented. The latter is also a very important part in the design flow, since after manufacture of an ASIC there is little or no possibility for correcting design errors (this is not the case with an FPGA).

Verification of an ASIC design starts early in the design process, i.e. in parallel with the creation of the design. The verification plan is an outcome of the *Architectural Design* phase, and ensures that the design is verified to a satisfactory manner and that the functionality is maintained throughout the subsequent design transformation steps. Failures during verification shall result in corrective actions dealing with the failure by modifying the HDL code or modifying the transformation tools scripts if those are the cause of the failure. In the design flow description (section 7.2) there are several verification steps between design transformation phases from HDL to the prototype chip. Some of these steps are formal, proving that the behaviour is maintained after the transformation, others rely on simulation of various functional scenarios in a test bench.

The development flow of HDL is quite similar to that of software. It includes top level system design from which the chip design specification is extracted. The chip is then designed in a top down manner, first defining the architectural breakdown, continuing with module design, and finishing with the actual module implementation and integration. Each block is then verified before being incorporated into the chip architecture and tested on chip level. The development follows the software development V-model, in the top-down designing and bottom up verification. But since the testing and verifying a HDL design is not a trivial task, especially when the target is an ASIC, it is important to start developing test benches early in the design process, such that the whole design process is not delayed or compromised by lack of test. It is not uncommon that much more than 50% of the total design time is spent on developing test cases and test benches. The emphasis on early test and verification development is shown in the modified V-model shown in Figure 11 where the test bench development is done in parallel with the design development itself. Also see [4], clause 7.1.2.4, which states that the designer of software may tailor the depth of the V-model provided that the software is developed and fulfils the extent of the software life cycle.

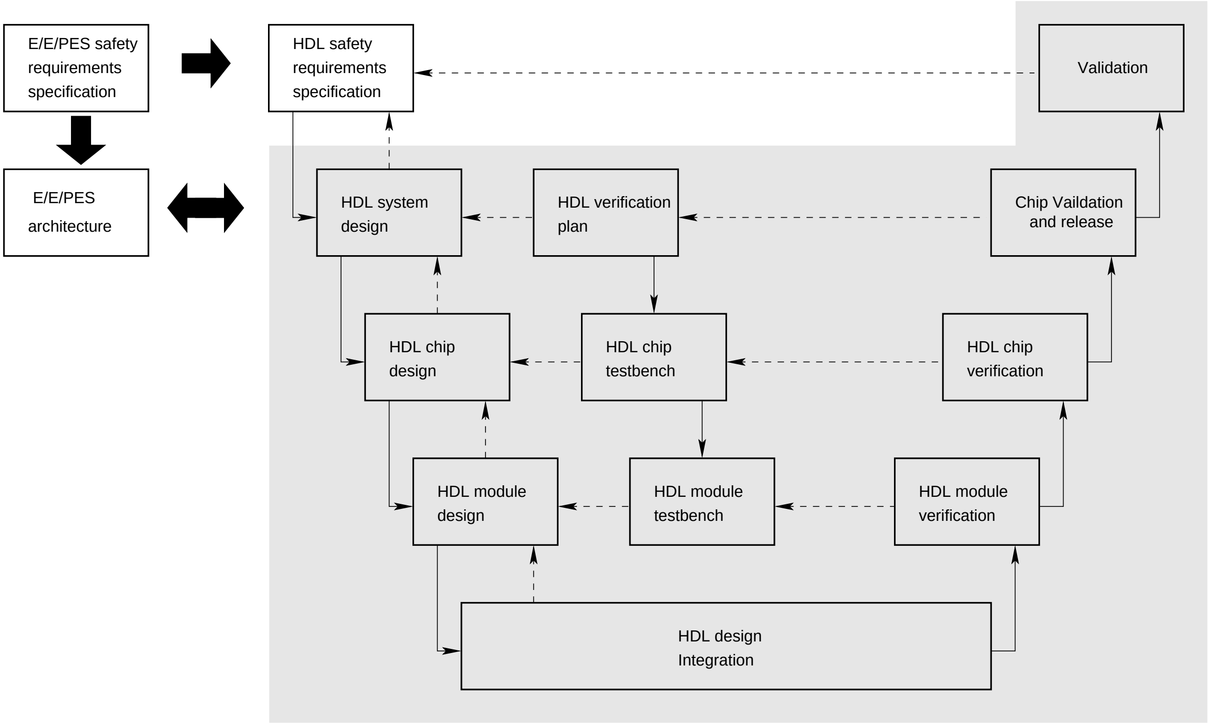


Figure 11: Modified V-model.

As described in section 7.2.3, there exist various techniques to test the design. Some performs a formal proof of functionality others simulate behaviour on input stimuli.

8.1.1. Formal verification methods

Formal methods are applicable for proving that all functionality has been maintained during the design transformation. Some of these are:

a) Static property check: makes a formal proof to verify that the signal properties are correct for all possible input sequences. This is still a new technology, but carries high expectations for the future. The goal is to specify the circuit behaviour in terms of signal behaviour through a signal property specification. The compliance of the design with the signal property specification can be verified formally. Ideally the signal property specification could make the link between the design specification and HDL, but the signal property specification cannot replace the design specification. It can only complement it.

b) Formal equivalence check: makes a formal proof that the RTL HDL has been transformed correctly. The RTL and the gate netlist are checked for full logical equivalence. The successful usage of the formal equivalence check relies on a correct RTL HDL design.

c) LVS (Layout versus schematic): compares the layout primitives with those found in the gate netlist. This check ensures that the gate netlist is fully implemented by the layout. Successful use of LVS requires a correct gate netlist.

d) Static timing analysis: analyses that all timing requirements for all gates in the design are fulfilled. This check requires that the timing for wires between gates are modeled, either extracted from the layout or estimated.

An implementation of a verification flow using the above techniques ensures the RTL to layout transformation quite well. However, the first link from the design specification to the RTL is the weak point, as it relies on human interpretation of the specification, either as a signal property specification or as test benches. This is why the verification plan must focus on how to test and verify the RTL at a satisfactory level.

8.1.2. Dynamic verification methods

Apart from the static property check some methods are:

a) Simulation: test cases and test benches are developed to test the design functionality. This should simulate the reaction of the design to correct and wrong inputs, thus simulating both correct behaviour and error handling.

b) Dynamic property check: uses the signal property specification to assert a warning if a signal property has been violated. Relies on HDL test bench.

c) Random test bench: generates random input (within some constraints) and simulates it with the design. This is used to stimulate the design with more or less obvious data to reveal well hidden bugs. This makes the creation of RTL test benches and test data easy, but the debugging of a possible failure can be quite difficult. This method verifies the design stability and robustness but does not directly verify its correct behaviour.

d) Fault injection: injects an error in a signal (change the signal value) and simulate its consequence. The method tests the robustness and failure immunity of the design, but should only be applied in areas where this is an important feature.

e) Code coverage: characterises the completeness of the RTL test bench by counting the number of times selected lines of code or constructs have been visited when running the test bench. Gives a statistical measure of how well the test bench simulates the design.

f) Semi formal methods: based on coverage measures, formal methods are used to try to activate difficult areas of the design. This method relies on coverage metrics, which could be obtained from the test benches or from experience.

g) Trial FPGA implementation: enables real time running of the design in its environment. Especially nice when the target is an ASIC.

8.1.3. Detailed verification plan

It is apparent that the creation of proper test benches is crucial to ensure a correct design, which demands a well prepared verification plan. This plan could cover:

a) Detailed specification of the test cases and test benches that should be run to verify the correct behaviour of the design on normal input stimuli.
b) Detailed specification of the test cases and test benches that should be run to verify the correct behaviour of the design on erroneous input stimuli.
c) Detailed specification on how the safety critical parts of the design are verified.
d) How the design should be tested, i.e. the mix of test benches using different methods.
e) Which tests should be done and at which level. Run time grows rapidly when performing gate netlist level simulations.
f) When is the design tested well enough. A design can be tested infinitely, but when is the test satisfactory.
g) How to handle the possible interaction with application software.
h) How to model the environment and interactions.
i) Plan for review of test benches.

The test benches have to be reviewed just like the design itself. Much of the behaviour of the design is built in the test benches, and it is equally important that the functionality of the test benches is as correct as the functionality of the design. For the design validation it is as important that the design and the test benches are well documented and delivered for control. A "what-if" analysis is far easier to perform through an established test bench than by reading the HDL code. Thus the test benches are in fact a part of the design, not an appendix.

8.2. Requirements on documentation

Structured HDL and ASIC/FPGA development demand structured documentation. One established way of structuring the design documentation is described in [1]. Here the main objective is to record the design development as it progresses through the design phases, and eventually get a design documentation database that contains a design description along with justifications for all design decisions made during the design phases.

The design phases contribute with different documents, starting from the requirements specification and the risk analysis via the HDL and the transformation tool scripts to the layout verification report. The following sections give a short description of documents that can be extracted during the ASIC design process presented in Figure 7 and Figure 9.

8.2.1. Architectural design

The main documentation from this phase consists of the design broken down to module level and a description of interfaces between modules. Decisions on how to implement algorithms and optimize them should also be covered. The verification plan is made and evaluated. The design results from this phase are:

a) Verification plan: How the design should be verified throughout the design phases.
b) Signal Properties: High level behaviour of selected signals
c) Test bench: Documentation on setup and on how to operate the test benches.
d) Simulation Model: Behavioural description of the design
e) Simulation: Results of the simulations

8.2.2. Detailed design

This phase mainly consists of transforming the design from a high level description to a netlist description. This involves a number of automated transformations, which all include scripts for controlling the transformation. The work and decisions performed during this phase must be documented. After transforming the design it must be verified. The result of these verifications must be documented. The results from the process steps during this phase are:

a) Simulation: Result of simulating the RTL model
b) Formal Verification (Property Check): Formal proof of conformance with the signal property specification
c) Synthesis: Control script containing decisions and method, and log file containing transcript of all operations
d) Design for Test: Control script and test solution decisions.
e) Re-Optimization: Control script containing decisions and method, and log file containing transcript of all operations
f) Static Timing Analysis: Timing report
g) Gate level simulation: Simulation results of gate netlist
h) Formal Equivalence Check: Equivalence proof between RTL model and gate netlist.

8.2.3. Layout design

As for the previous phase most of the transformations in this phase are automated and script controlled. The documentation will therefore contain a description of the performed work, decisions taken, and report the results of the verification. The results gathered during this phase are:

a) Floor planning: Report of decisions and flop plan strategy for the chip layout.
b) Sub-block Place & Route: Control scripts and area and timing report in log file
c) Chip level Place & Route: Control scripts and area and timing report in log file
d) Layout Verification: Design rule check report and "Layout vs. Schematic" validation report
e) Static timing analysis: Timing report
f) Test Pattern Generation: Production test fault coverage
g) Layout Level simulation: Simulation result of layout gate netlist
h) Formal Equivalence check: Equivalence proof between RTL and layout gate netlist

For more information about system verification refer to e.g. [19]

8.3. Requirements on safety validation

This clause contains a brief introduction to some commonly used validation methods. None of the analysis methods mentioned are compulsory but they are all strongly recommended. It is assumed that the design process described in section 8 of this report has been followed and that all required documentation has been produced.

The design process described in section 8 shall be verified both for compliance with the requirements listed in this report and with all additional relevant requirements listed in standards [1] and [2]. The validation shall furthermore be carried out in accordance with the validation plan produced during the design process.

Following documents are particularly important:

-E/E/PES Safety requirements specification, that includes:
-E/E/PES Safety functions specification
-E/E/PES Safety integrity requirements specification
-E/E/PES Safety validation plan

The validation methods mentioned in this section are not trivial in practice and there are several cases and system configurations that require a deeper understanding of these methods than mentioned here. When validating/verifying a real E/E/PES most of the described methods require computer aid.

Not all recommended validation methods and analysis techniques are addressed in this clause, for further information about other suitable methods see [7].

When performing the safety validation the below requirements shall be considered:

8.3.1. Requirements on E/E/PES safety validation

The objective of safety validation is to validate that the system fulfils the safety requirements for safety functions and safety integrity levels, entirely. The validation shall be carried out according to the validation plan. All used measurement instruments shall be calibrated against a traceable standard or a well-recognized procedure. The instruments must also be verified for correct operation prior to use.

Each E/E/PES safety function specified in the safety requirements specification shall be validated by test and/or analysis. The E/E/PES maintenance and operation procedures shall also be validated.

For each safety function validated the following documentation shall be produced:

a) Version of the safety validation plan
b) The tested (or analyzed) safety function with reference to the safety requirement specification
c) Tools and equipment along with calibration data
d) Test results
e) Discrepancies between expected and actual results. Following additional documentation shall be added for the discrepancies:
f) The analysis made
g) Decision made on continuing test or issuing a modification request

The developer of the EUC and the EUC control system shall have access to the results of the performed safety validation. In order to avoid faults during the performance of the validation [3], table B.5 should be used.

If a deviation that requires a design modification is detected during the validation the requirements listed in clause 8.3.2 below must be considered.

8.3.2. E/E/PES Modification

The required safety integrity of an E/E/PE safety-related system shall remain after corrections/modifications. Regarding ASIC-based systems this is an important matter since such systems often are ISP.

[3] requires the following documentation after a system modification activity:

- Detailed specification of the system modification
- Analysis of the modification impact on the overall system (e.g. hardware, software, human interactions)
- All approvals for changes
- Progress of changes
- Test cases for components including revalidation data
- E/E/PES configuration management history
- Deviations from normal operations and conditions
- Necessary changes to system procedures
- Necessary changes to documentation

Manufacturers or system suppliers shall maintain a system in order to inform the users of detected defects affecting the safety. Modifications shall be performed with at least the same level of expertise, automated tools, planning and management as the original design for fulfilment of IEC 61508.

After modification the E/E/PES safety-related system shall be reverified and revalidated.
Ref. [3], clause 7.8

8.4. Methods for static analysis

Static analysis techniques aim to theoretically prove that particular requirements are fulfilled or point out the parts of the design that shall be subject to further dynamic analyses and functional tests.

8.4.1. FBA (Functional Block Analysis)

Aim:
To extract the safety-related parts of the subsystem and to provide the analyst with knowledge on how different parts (modules) of a subsystem interact.

Performance:
The FBA method produces a graphical presentation of the parts of the subsystem subject to further detailed analysis. This method is usually the first to be performed in a safety validation. Several persons may/should be involved in the analysis performance process. The performer(s) of the analysis may select different graphical representations and/or system levels depending on what kind of system is analyzed.

Result:
The result of the method is not necessarily restricted to either software or hardware. It shall rather give an overview of the complete function. Great care should be taken when performing the FBA as the result will not only incorporate safety-related parts of the subsystem but will also exclude parts not considered to be safety-related. It is important that no part of the subsystem that may affect the safety function is excluded.

One method of analyzing HDL is to use equivalent hardware circuit diagrams. This method has several advantages:

- The process of drawing the result of FBA is often automated (i.e. the possibility of illustrating the VHDL code as a circuit diagram is commonly integrated in the development tool)
- The ASIC under consideration will be transparent due to its surrounding electronics
- The FBA will clearly display the level of separation (redundancy) between channels
- The FBA will prepare for other static analysis methods such as data flow diagrams and FMEA
- The FBA may be hierarchically presented. This is necessary for more complex systems

Applicability:

- All system levels in the hardware platform
- System levels in the HDL description covering the RTL-description and higher levels.

Example:
See appendix B figure B.2 and B.3 which both are different kinds of FBAs

See [7], Annex B.2.3.2

8.4.2. DFA (Data flow Analysis)

Aim:
To determine exactly which parts of the system affecting a safety-related data path and which data variables/signals are involved in the occurrence of a single safety-related event resulting from the alternation of the system input signals.

Performance:

A graphical representation of the data flow from input signals to safety-related output signals is produced. The data flow analysis is independent of the conditions involved in the chain of events but considers only information carrying signals. One safety function usually contains several data flows.

Result:
The analysis technique gives information about e.g.

- Information stored in the system which affects the safety function
- Implemented data diversity/redundancy
- Allocation of safety-related information in the system
- Conformance with the system specification

Applicability:

- All system levels in the hardware platform
- System levels in the HDL description covering the RTL-description and higher levels.

See [7], Annex B.2.3.2

8.4.3. SSA (Signal Sequence Analysis)

Aim:
To determine the logical conditions and required sequence of events that enable input signals to affect a safety-related output; similar with the control sequence analysis described in [7].

Performance:
The HDL code and the detailed result of the FBA are studied and a graph is produced with emphasis on showing all conditions that have to be fulfilled before any input is allowed to affect a safety-related output. By manually parsing the HDL and interpreting the code as circuit primitives (such as gates, multiplexers, registers) the code may be drawn as a circuit diagram that displays the functionality at the level of granularity required for the analysis purpose. Only specifically interesting parts of the design should be considered.

Result:
A graphical presentation of the relationship between different signal conditions used to illustrate and clarify specific safety-related sequences of events and condition consistency.

Applicability:

- This analysis technique may be used for parts where it is not more illustrative to use STDA (See section 8.4.4).
- All system levels in the hardware platform
- System levels in the HDL description covering the RTL-description and higher levels.

See [7], Annex B.2.3.2

8.4.4. STDA (State transition Diagram Analysis)

Aim:
To analyze a sequence performed by the system focusing only on the system conditions for state transitions due to safety-related signals regardless of the source of individual signals.

Performance:
Studying the safety-related parts of the control system and constructing a graphical state transition diagram that includes all safety-related states and all other states that have the possibility of transiting to the safety-related states. The state transition diagram shall model the system control structure. A recommendation is to apply the technique at a sufficiently high system level.

Result:

- The order of interlocking events for initiating a safety-related action
- The conditions for the above mentioned events
- Possibilities of entering fail-safe states

This analysis technique is very suitable for HDL realized subsystems as any sequence process will be implemented as a physical state machine in the hardware.

Applicability:

- All system levels in the hardware platform
- System levels in the HDL description covering the RTL-description and higher levels.

See also [7], Annex B.2.3.2

8.4.5. FMEA (Failure modes and Effects Analysis)

Aim:
To identify possible sources of failure in the system components and to determine the consequences in terms of system behaviour due to the occurrence of these failures.

Performance:
Failure modes and effects analysis is a bottom-up method that analyses potential failure modes and their causes and effects on system performance. To be able to perform an FMEA a predefined fault model has to be selected. The fault model defines how the system elements may fail in operation and how extensive the FMEA will become. The FMEA is not necessarily restricted to electronic hardware. It may be apply as well on e.g. firmware, communication protocols, mechanic systems, pneumatic systems or hydraulic systems.

8.4.5.1. Defining the scope of the analysis
Because the FMEA is a detailed analysis it usually becomes large. It is therefore necessary to extract only the parts of the system that are relevant for the analysis. The easiest way is to create a functional block diagram of the system and reduce the non safety-related blocks (FBA). The functional block diagram gives an overview of the system at a high system level. It is important to carefully study all interfaces between the blocks so that no safety-related signal is forgotten and thereby excluding a safety-related part from the analysis. The result of the functional block diagram reduction should be documented and motivated.

8.4.5.2. Determining the system success/failure criteria
The requirement for fault tolerance is defined by a product standard or as a result of a risk analysis and is often very general. It is recommended to translate the requirement into a criterion that is precisely defined for the actual safety function(s) in the system to be analyzed. It is also important to distinguish the different operational states of the system and how the requirement(s) applies(y).
Examples of operational states:
Start-up, operation, shut-down, degraded-operation, safe-state, and safety shut-down.

8.4.5.3. Defining the failure model
The following aspects should be considered and documented before engaging the analysis:

- The environment in which the system is to be used
- The failure mode behaviour with respect to time (transient, intermittent, permanent)
- The type of failure mode to be analyzed (random single failure, common cause failure, differential cause failure)
- The applicability of failure modes (may some failure modes be excluded? This is often mentioned in the product standards or component specific data sheets)

The failure model does not necessarily concern single components. An FMEA may as well be performed at a higher system level using failure modes that involve a group of components within subsystems.

8.4.5.4. FMEA worksheet

The FMEA is performed by using a worksheet. The following example illustrates how an FMEA worksheet may be formed:

Table 9: FMEA example worksheet

Ref.	Component	Failure mode	Local effect	System effect	Failure detected?	Failure dangerous?
1	R11, 10k 5W	Short	Motor overload protection disabled	Unintended motor stop	Yes (motor overload monitor/protection)	No
2	Push-button SW3	Short	CPU receives start command	Unintended motor start-up	No	Yes

Each component/element included in the FMEA is listed with its corresponding failure modes and effects. For more detailed information about the FMEA worksheet see [12].

Result:
The result of the FMEA is especially used when claming conformance with the requirements in IEC 61508 and is compulsory to determine the DC and SFF.

Applicability:

- All system levels in the hardware platform
- System levels in the HDL description covering the RTL-description and higher levels.

See also [7], Annex B.6.6.1

8.5. Methods for dynamic analysis

Aim:
Prove that particular requirements are fulfilled by testing specific properties of a physical prototype or product. These techniques are used for the validation of a complete subsystem to show that the functional requirements concerning the system properties are fulfilled. The methods and techniques in this section are referred to as FIT (Fault Injection Testing) and SPT (System Property Testing) in section 8.8.

Performance:
Dynamic method performance depends on the tool in use.

Result:
Evidence that the requirements for the system behaviour or system properties are fulfilled.
The result is limited to the extent of the specification of the actual dynamic test procedure.

Applicability:
These methods may apply during the design process (verification) but primarily on an end-line prototype.

Example: Verifying that the time-requirement is still fulfilled regardless of an applied failure mode derived from a previous FMEA.

See also [7], Annex B.6.5

8.5.1. Simulation/Test benches

The VHDL design tools provide powerful means for simulating operation of all parts in a VHDL design and are therefore suitable in most stages of the design process. One example is the verification of state transitions in the V-model. Such tests shall be planned and performed according to the requirements in section 8.1.

Test benches are VHDL models that act as the circuit surrounding electronic in the VHDL simulation tool and that generate input signals and handle the corresponding output signals. These are very useful in order to verify functional compliance with the system specification. The part of the design subject to the test bench is, from the test bench point of view treated as a black box. The detailed test bench design should be planned concurrently with the design.

- The verification result depends on the test bench design
- The results are retrieved from the simulator "*ideal*"-environment which may lack influential factors from the physical environment (such as at-speed-failures, specific environmental stresses or degradation in the circuit construction process)

8.5.2. ASIC Emulation

ASIC emulation is similar to microprocessor emulation and is performed by applying an emulator pod to the application hardware platform and connecting that pod to a VHDL simulator environment. The emulation is a circuit simulation where the test bench in fact is the physical application platform.

8.5.3. Fault injection

Fault injection is a means of failure analysis where the fault is injected into the hardware and the consequence of the failure is directly analyzed in operation. There are several different means for injecting failures into the design and then analyzing the result of the failures. Different types of fault insertion are presented in the following sections.

8.5.3.1. Design

The fault is injected by design modifications, e.g. forcing a signal to a certain value. The consequence of the fault is analyzed by simulation, emulation or during normal operation (when using ISP programmable logic). It is important to correctly restore the design after having analyzed a fault. This way of inserting the fault is passive. The inserted fault is a permanent fault.

8.5.3.2. Saboteurs

A saboteur is a more or less complex device that injects a fault as a function of its input parameters, i.e. the fault injection is conditional. Examples of saboteurs may be:

- Replacing a certain package in a communication process with an faulty package
- Forcing the value in a register at a certain time
- Varying the persistence and moment of occurrence for a single fault

The saboteur may also be connected to the test bench and hence allowing automated parametrical fault injection. When using saboteurs the following should be considered,

- The effect on the normal operation of the system shall be minimized (e.g. a saboteur may prolong the delay-time and hence faulty enhancing the system ability of detecting the fault injected)
- It is important that the saboteur is correctly removed from the design after use. The injected faults from a saboteur may be permanent, intermittent or transient and affect all functional system levels of the design.

8.5.3.3. Environmental influences

Additional to the required environmental tests further tests may be used for fault injection. Examples of environmental fault injection may be:

- Transient voltage injection on the system inputs (Power supply, Clock signal, I/O etc.)
- Heavy ion injection (injects a large amount of stochastic single point failures at a given time period)

These methods require a sample of the implemented system and the tests are not applicable if using e.g. a prototype implemented in a fine grained FPGA. The repeatability of these tests is low for detected failures and it is difficult to localize the fault that causes the failure.
These methods should primary be used on designs in which the safety function depends on the ASIC and as complement to all other methods mentioned in this report.

8.6. Methods for reliability analysis

Methods for reliability analysis are a subset of static analysis methods and aim to predict or estimate quantitative measures of the reliability of a system, subsystem or component.

Aim:
To estimate the probability of failure for a system, a subsystem or a component.

Performance:
Safety-related parts are distinguished from the design. A reliability measure is retrieved for each component (which may be subsystems) in the safety-related part and by using the methods below the resulting reliability measure is estimated for the complete safety-related part.

Result:
One single reliability measure for a subsystem or safety-related part.

Applicability:
Subsystem level; it is not possible to perform trustworthy reliability estimations within an integrated circuit.

References: [9], [11], [13]

8.6.1. Reliability Block Diagram (RBD)

The reliability block diagram aims to describe a system function in terms of the reliability of the system elements composing that particular function. The reliability block diagram describes the function by connecting its elements, usually in series, in parallel or using a combination of components series and in parallel. A reliability measure is assigned to each element. The RBD is eventually used for rendering a reliability measure for the complete function.

All elements are assumed to be completely independent of each other. Failures or repairs to individual blocks are considered to be statistically independent events. The RBD is not suitable for modelling order or time dependent events; compare other modelling techniques such as Markov analysis for such cases.

The RBD does not necessarily represent the physical hardware of a system but the relationship between different functional elements that combined form a system function. Each element in an RBD represents substructures that may represent another RBD (system reduction).

In order to construct the RBD the following topics have to be considered:

- The system fault definition (e.g. FMEA)
- Performance parameters and permissible limits of such parameters
- Environmental and operating conditions
- Duty cycles

8.6.1.1. RBD model evaluation

Consider appendix A that states the relationship between the reliability function R and the unreliability function Q so that R = 1 – Q

Series models:
The resulting reliability is given by multiplying the individual reliability of all elements that contribute to the realisation of the function. The function will operate success only if the reliability is > 0 for each element at a given time.

$$R_{tot} = \prod_{N=1}^{i} R_N$$

where N is the number of elements in series and R_N is the reliability for each element.
$R_{tot} = R_1R_2$ for N = 2

Parallel models:
The resulting unreliability is given by multiplying the individual unreliability of all elements that contribute to the realisation of the function. The function will fail only if the reliability = 0 for each element at a given time.

$$Q_{tot} = \prod_{N=1}^{i} Q_N$$

where N is the number of elements in series and Q_N is the unreliability for each element
This may be expressed in terms of reliability using the relationship R + Q = 1

$$R_{tot} = 1 - \prod_{N=1}^{i}(1 - R_N)$$

$R_{tot} = R_1 + R_2 - R_1R_2$ for N = 2

More complex models
If the RBD cannot be expressed only as a combination of elements in series and in parallel another approach has to be used.

Using the conditional probability rule
Rs = Pr(SS/X operational) x Pr(X operational) + Pr(SS/X faulty) x PR(X faulty)

Rs denotes the system reliability
Pr(SS/X operational) – The probability of system success if element X is operational ($R_x = 1$)
Pr(SS/X faulty) – The probability of system success if element X is faulty ($R_x = 0$)

When using this formula the RBD is calculated in steps in order to retrieve the above described parameters. The formula is then applied to retrieve R_s.

Using Boolean truth tables
Any RBD may be displayed in a truth table where all blocks are represented as either operational (r_n) or faulty (r'_n) and in all possible combinations. The table will contain N columns and 2^N rows, if N is the amount of elements in the RBD. The resulting expression is formed by the sum of products of each row that makes the system success (SS) become true.

Example of a simple RBD:
This example illustrates how the described methods for evaluation may be used.

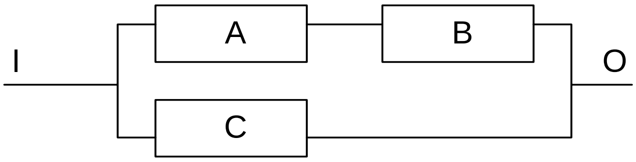


Figure 12: Simple RBD example

Using series and parallel model expressions,
A in series with B ⇔ $R_{AB} = R_A R_B$
C parallel to R_{AB} ⇔ $R_S = R_{AB} + R_C - R_{AB}R_C = R_A R_B + R_C - R_A R_B R_C$
The resulting system reliability from Figure 12 becomes: $R_S = R_C + R_A R_B(1-R_C)$

Using a truth table

Table 10: Truth table example

A	B	C	SS
0	0	0	F
0	0	1	T
0	1	0	F
0	1	1	T
1	0	0	F
1	0	1	T
1	1	0	T
1	1	1	T

Rewritten in a Boolean expression the system success formula becomes,
$SS = r'_A.r'_B.r_C + r'_A.r_B.r_C + r_A.r'_B.r_C + r_A.r_B.r'_C + r_A.r_B.r_C$

Using basic Boolean algebra this formula may be minimized to:
$SS = r_C + r_A.r_B.r'_C$

By inserting the reliability terms the system reliability formula is created,
$R_s = R_C + R_A R_B(1-R_C)$

Substitution of probability variables by Boolean variables
Instead of using truth tables it is sometimes more suitable/convenient to use general Boolean algebra for such analyses. The algebraic method is the most straight forward method when

- the RBD contains several similar elements
- the RBD contains directional relations between elements
- the RBD is complex

It is although important to keep following in mind when using Boolean algebra:
A system contains two redundant subsystems A and B with the Boolean survival variables a and b. The system success may be expressed in Boolean form as,
$SS = a + b$

The variables are overlapping each other and therefore this relation cannot be substituted directly by probability variables as previously when using the truth table method (The result would incorrectly be $R_s = R_A + R_B$)

In order to get a correct result the Boolean form SS = a + b first has to be re-written in order to ensure that the variables are not overlapping. This process is called disjointing and after the process has been performed no term in a sum of products overlaps another term. One way to disjoint the example above is:
SS = a + a'.b

After substituting the probability variables into this rewritten expression:
$R_s = R_A + (1\text{-}R_A)R_B$ which is correct.

Each term of a Boolean expression has to be disjointed with respect to every other term. Two terms are mutually disjointed if at least one variable in one term appears in its complementary form in the other term.

Disjointing a Boolean sum-of-products with two terms:
Assume two terms T_1 and T_2. In order to disjoint T_2 with respect to T_1 use the following steps,

a) Retrieve all variables from T_1 which do not appear in T_2 (assume v_1,v_2,v_3,v_4)
b) Expand T_2 with $T_2* = v'_1.T_2 + v_1.v'_2.T_2 + v_1.v_2.v'_3.T_2 + v_1.v_2.v_3.v'_4.T_2$
c) Resulting sum-of-products: $T_1 + T_2*$ where the two terms are disjointed with respect to each other

Procedure for a complete sum-of-products:
Assume $SS_1 = T_{11}+T_{12}+..+T_{1n}$, T_{11} is the reference term

1) Disjoint with each term T_{12} .. T1n as described above ($T_{11} - T_{12}$, $T_{11} - T_{13}$, .. , T_{11}- T_{1n})
2) Simplify the resulting SS_1 but keeping the sum-of-product form: $SS_2 = T_{21} + T_{22} + .. + T_{2n}$
3) Repeat the procedure 1-2 using on SS_2 using T_{22} as the reference term (T_{21} is now disjointed with the other terms and may therefore be left out)
4) Continue 1-3 until all terms are disjointed, and the result is $SS_n = T_{n1} + T_{n2} + .. + T_{nn}$

The following example shows how the disjointing procedure may be manually performed. When a Boolean expression is larger it may be suitable to implement the procedure as an algorithm in a computer.
Assume that the following Boolean expression is extracted during an analysis,
SS = a.b + b.c + d where T11 = a.b, T_{12} = b.c, T_{13} = d

Step 1.1
SS_1 = SS
First handle T_{11} and T_{12}.

a) Find variables in T_{11} that do not appear in T_{12}: a
b) $T_{12}* = a'.T_{12} = a'.b.c$

Handle T_{11} and T_{13}

a) Find variables in T_{11} that do not appear in T_{13}: a.b
b) $T_{13}* = a'.T_{13} + a.b'.T_{13} = a'.d + a.b'.d$

Step 1.2
And hence SS_1 becomes:
SS_2 = a.b + a'.b.c + a'.d + a.b'.d, where T_{21} = a.b, T_{22} =a'.b.c, T_{23} = a'd, T_{24} = a.b',d
The first term T_{21} is disjointed with all other terms.
Handle T_{22} and T_{23}

a) Find variables in T_{22} that do not appear in T_{23}: b
b) $T_{23}* = b'.T_{23} = b'.a'.d$

Handle T_{22} and T_{24}

a) Since T_{22} contains b and T_{24} contains b' they are already disjointed
b) $T_{24}*$ = T24

Step 1.3
Hence SS_2 becomes:
SS_3 = a.b + a'b.c +a'.b'.d + a.b'.d, No pairs of terms lack a common variable that is not complementary and therefore is all terms are disjointed and ready for substitution with probability variables.
After some simplification of SS_3 the final result becomes,
SS = b(a + a'c) + b'd

When performing this procedure, one should always range the sum-of-product so that the term with the lowest number of variables is used as the first reference term (T_{11}) and the term with highest number of variables as the last (T_{mn}).

Comments about disjointed Boolean expressions:

- The result of the method allows substitution of probability variables (reliability variables)
- The method may be efficiently used in Boolean expressions arising from Fault Tree Analysis
- Instead of substituting reliabilities by the Boolean variables, availabilities can be used

For more information about this topic, see [11].

8.6.2. Fault Tree Analysis

The fault tree (FT) analysis technique is a method for top-down analysis that is suitable for both qualitative and quantitative analysis of E/E/PE systems. The FT displays the relation between a top-event and its intermediate events. Logical gates exclusively determine the relationship between these events. All events in the FT are required to be independent of each other. The FT method is a cause-consequence process and when applied on E/E/PE system reliability it becomes an efficient tool for analyzing the E/E/PE system provision for failure and risk reduction.

In the process of developing the FT the top event (basic system failure) becomes divided into its intermediate events (lower system level faults) and their corresponding relationships. The granularity of the FT is then refined and expanded until the lowest desired system level is reached. The final events are denoted *basic events*, which also define the *limit of resolution* for the entire FT.

In the figure below some basic FT symbols are viewed. For further information about other symbols, see [13].

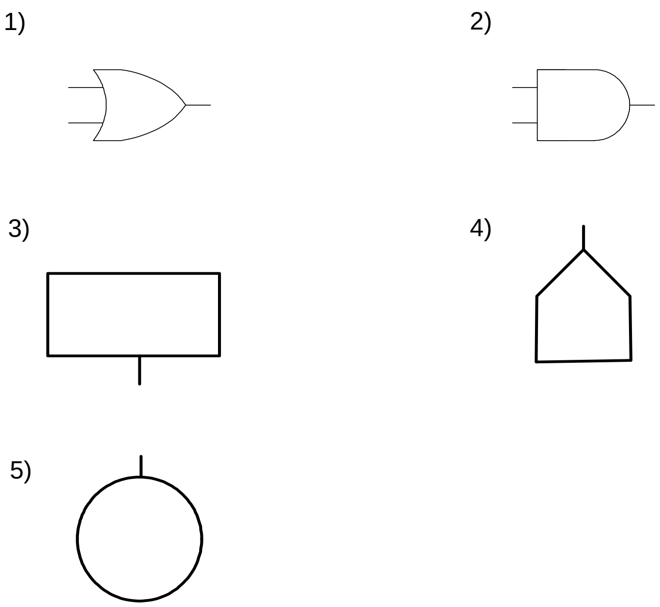


Figure 13: Basic FT symbols.

Figure 13 Symbol 1) – The OR-gate. This gate is used when a failure (event) is occurring if any of the intermediate failures (events) occurs.

Figure 13 Symbol 2) – The AND-gate. This gate is used when a failure (event) occurs only if all intermediate events occur.

Figure 13 Symbol 3) This symbol illustrates the top event or intermediate event. The interpretation of such an event may be e.g. the failure of a system, the failure of a subsystem or function or the failure of a component depending on which level in the FT the symbol is present.

Figure 13 Symbol 4) This symbol is referred to as a *house event* which means that this particular event is expected to occur. The house event does not necessarily have to be the consequence of a fault.

Figure 13 Symbol 5) This symbol illustrates the *basic event* which also defines the limit of resolution. The interpretation of these events also differs depending on the system level at which they are used, e.g. short in a resistor, failure to generate correct PWM-signal, source code execution fault etc. In order to enable quantitative analysis the probabilities for these events have to be retrieved from an external source.

When the FT is finished it may be used for quantitative analysis. This is done by forming a Boolean expression where the basic events are the Boolean variables. In order to replace the Boolean variables with probability variables refer to section 8.6.1.1 substitution of probability variables into Boolean variables.

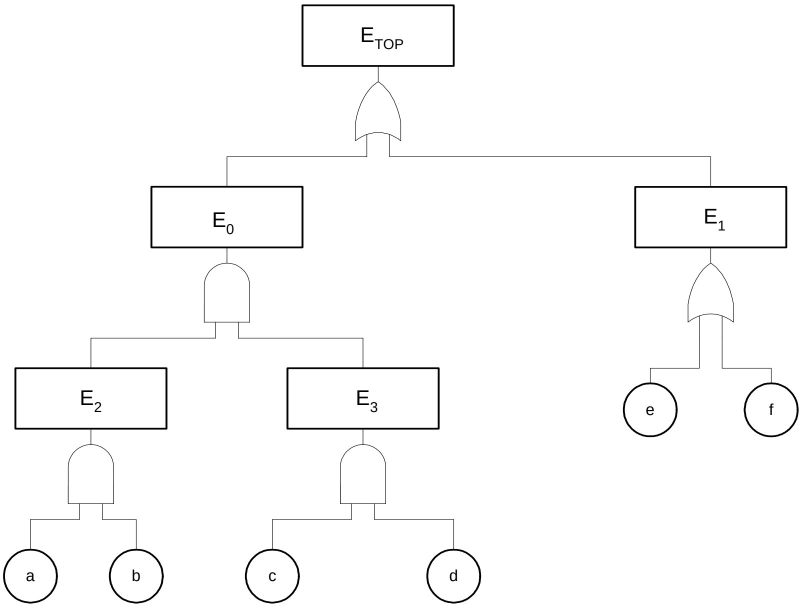


Figure 14: Fault tree example.

To render a Boolean expression for the FT example in Figure 14,
$E_{TOP} = E_0 + E_1$, $E_0 = E_2.E_3$, $E_1 = e+f$, $E_2 = a.b$, $E_3 = c.d$

$E_{TOP} = a.b.c.d + e + f$

In order to perform quantitative analysis, the Boolean expression must first be disjointed. The result of the disjointing procedure is,
$E_{TOT} = a.b.c.d.e'.f' + e'.f + e$

8.6.3. Markov chain modeling

The Markov chain considers a sequence of events and analyses the transition paths from any state to another. There are two basic Markov analysis methods, the Markov chain
(Discrete states and discrete time domain) and the Markov process (continuous states). A Markov chain may be described as:

- homogeneous, with constant transition rate between states or
- non-Homogeneous, where all transition rates are functions of a global clock (e.g. elapsed mission time).

The Markov model analyses the probability of the transition between one known state (*i*) to the next state (j) where *i,j = 1,2,3…n* . Any transition is assumed to be completely independent of the previous states and the time of occurrence of the previous transition.

In terms of reliability Markov modelling may be used to analyze the probability of system failure. The basic principle of the Markov chain is displayed in the example below,

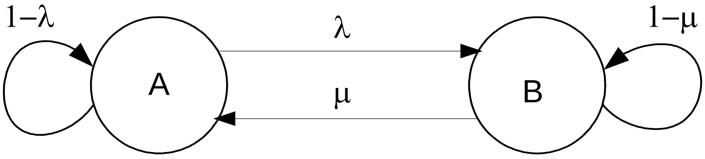


Figure 15: Simple Markov model example

The Markov chain in Figure 15 exemplifies an E/E/PES where the system is either in the state A – operational or in the state B – failed. The system will remain in the state A until – Operational until a failure occurs and results in a transition to the state B - failed. The transition between A and B is determined by the failure rate λ(t). The system will remain in the state B - failed until it is repaired and hence re-transit to the system operational state A. The transition between B and A is determined by the repair rate μ(t) .

The example above is coarse and while expanding the analysis to more detailed system levels the Markov chain will grow into a state transition diagram possibly including a huge amount of states and transitions.

The Markov chain is often described in mathematical terms as:

(E1) $$\tfrac{d}{dt}\underline{P} = [A] \cdot \underline{P}$$

where $\underline{P}$ is an *1 x n* column vector, [A] is an *n x n* matrix and n is the total number of states to be considered during the analysis.

The matrix [A] is obtained by examining the system due to each state n and the probability for the system of being in each of the state n at a time t + Δt as a function of the current state of the system at time t.

Consider for example an E/E/PE system or subsystem providing a service that is composed of two separate functions. If one of these functions fails the system is degraded but still operational, if both functions fail, the service is disabled and detected by the system that hence becomes enabled for repair. The E/E/PE system is designed to minimize the common cause factor (β).

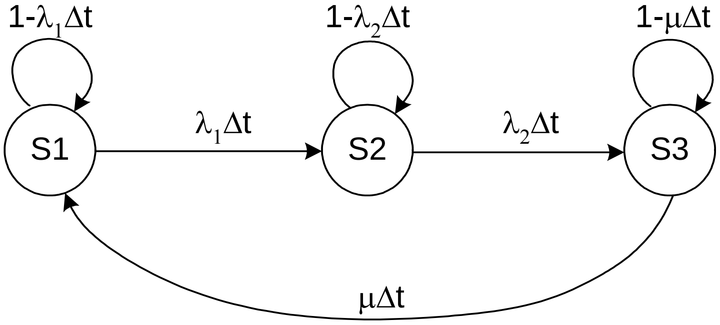


Figure 16: Example Markov chain.

S1 – The system is fully operational

S2 – The system is degraded in operation but remains to deliver the service. Since the failure does not affect the service, the repair rate from this state is considered as larger than the system life-time. The system cannot recover from this state.

S3 – The system is disabled and cannot deliver its service, the system will remain in this state until repaired.

In this example n = 3

The A matrix is composed of all the equations P_n for n = 1..3 (where P_n is the probability for the system of being in state n at the time t + Δt expressed as a function of the system state at time t) so that the probability of being in one of the states P_n at a given time t always equals 1.

The probability of P1 therefore equals the probability of not transit to P2 or being repaired and hence transiting from P3 during Δt. The probability of P2 equals the probability of either transit P1→P2 or not transit P2→P3 during Δt. The probability of being in P3 equals the probability of transiting to P2 → P3 or not being repaired during Δt.

$$P_1(t + \Delta t) = P_1(t)\,[1-\lambda_1\Delta t] + P_3(t)\,\mu\Delta t$$
$$P_2(t + \Delta t) = P_1(t)\,\lambda_1\Delta t + P_2(t)[1-\lambda_2\Delta t]$$
$$P_3(t + \Delta t) = P_2(t)\,\lambda_2\Delta t + P_3(t)[1-\mu\Delta t]$$

The above equations may be rearranged into,

$$[P_1(t + \Delta t) - P_1(t)]/\Delta t = -P_1(t)\,\lambda_1 + P_3(t)\mu$$
$$[P_2(t + \Delta t) – P_2(t)]/\Delta t = P_1(t)\,\lambda_1 - P_2(t)\,\lambda_2$$
$$[P_3(t + \Delta t) – P_3(t)]/\Delta t = P_2(t)\,\lambda_2 - P_3(t)\mu$$

When Δt → 0 in the equation above is recognized as (compare with equation E1)

(E2) $$\frac{d}{dt}\begin{bmatrix} P_1 \\ P_2 \\ P_3 \end{bmatrix} = \begin{bmatrix} -\lambda_1 & 0 & \mu \\ \lambda_1 & -\lambda_2 & 0 \\ 0 & \lambda_2 & -\mu \end{bmatrix} \cdot \begin{bmatrix} P_1(t) \\ P_2(t) \\ P_3(t) \end{bmatrix}$$

The equation (E2) usually has to be solved using special mathematical algorithms, especially when the number of states enlarges. However, computer aided tools for supporting Markov modeling with these algorithms implemented exist. The information to provide these tools with is the state transition diagram and all failure rates and repair rates.

The Markov modelling method is useful for analyzing the probabilities of sequenced events. Here are some disadvantages of Markov modelling:

- λ and μ may differ several orders of magnitude from each other.
- Enormous amounts of states if a too detailed system level is analyzed

For further information about Markov chain modelling please see e.g. [9], [11], [14] and [15].

8.6.4. Availability Analysis

The PFH of a system denotes the system unavailability which means the probability that the system is unable to deliver its correct service (i.e. the safety function) at a given point of time within the range of the proof test interval (mission time). See [11] for further information about availability.

MDT – Mean Down Time: The average time a system is unavailable due to failure.

Availability is applicable for repairable systems and is therefore applicable on a high system-level. The concept of uptime and downtime denotes the average time during which the system is available (uptime) within the system mission time and the average time during which the system is unavailable due to failure and repair (downtime). See Figure 17 for an illustration of the system availability over its mission time.

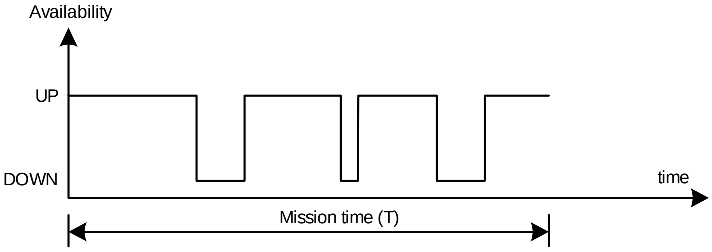


Figure 17: System availability due to the mission time.

The momentary availability of the system is expressed as A(t) which is the probability that a system is available at a given time t after start of operation. The mission availability is the average value of A(t) during a time range and is expressed as

$$A_{AV}(t_2 - t_1) = \frac{1}{t_2 - t_1}\int_{t_1}^{t_2} A(t)dt$$

where the time interval (t2 – t1) commonly equals the mission time T which denotes the proof test interval in IEC 61508 and the expression hence becomes:

$$A_{AV}(T) = \frac{1}{T}\int_{0}^{T} A(t)dt$$

The steady-state availability is retrieved when T is large or unknown and is determined by:

$$A_S = \frac{Limes}{t \to \inf} A(t)$$

The achieved availability/unavailability is determined by:

$$A_A = 1 - \frac{downtime}{T} = \frac{uptime}{T} \text{ or } U_A = 1 - \frac{uptime}{T} = \frac{downtime}{T}$$

When studying the availability of complex systems, Markov modelling is commonly used. Please refer to section 8.6.3 in this report for introducing information about Markov modelling. The result of the Markov model is a differential equation: $\frac{d}{dt}\underline{P} = [A] \cdot \underline{P}$ where P is the probability of being in a specific operational mode at a certain time.

Given an initial condition: $\underline{P}(t = 0) = \underline{k}$ this differential equation is solvable. If the matrix A is reasonably small the differential equation may be transformed into the Laplace domain using the Laplace transform. In this domain the equation may be solved explicitly as an ordinary system of equations. When the desired variables (target probabilities) have been extracted the equations are transformed back to the time domain using the inverse Laplace transform.

Availability analysis $A_{AV}(t)$:
Consider the average probability of remaining in an operating state where it continues to deliver its correct service i.e. $A(t) = P_{SUCCESS}(T)$. The mean value of A(t) is then calculated in order to get the steady state availability $A_{AV}(t)$.

$$A_{AV}(t) = \frac{1}{T}\int_{t_1}^{t_2} P_{SUCCESS}(t)dt$$

Unavailability analysis $U_{AV}(t)$:
Consider the average probability of entering an operating state where it is unable to deliver its correct service during its lifetime i.e. $U(t) = P_{FAILURE}(T)$. The mean value of U(t) is then calculated in order to get the steady state unavailability $U_{AV}(t)$

$$U_{AV}(t) = \frac{1}{T}\int_{t_1}^{t_2} P_{FAILURE}(t)dt$$

The availability/unavailability value obtained by solving the equations above is composed oftwo terms, one transient term and one steady-state term. If the mission time (proof test time interval) is very large the transient term may become negligible. It can be shown for a subsystem or an independent function that the steady state term always is expressed by:

$$A_S = \frac{\mu}{\lambda + \mu} \text{ or } U_S = \frac{\lambda}{\lambda + \mu}$$

If this approximation is trustworthy, A_S or U_S may be substituted directly into the resulting reliability expression from e.g. a reliability block diagram.

The value of the PFD_{AV} or the PFH_{AV} denotes the unavailability of the system $U_{AV}(t)$. The PFD_{AV} denotes the average probability of failure (i.e. the average probability for the system to enter one of the system degradation states in the Markov model during the mission time). The PFH_{AV} denotes the average probability of a failure to occur per hour (i.e. the average rate that the system enters one of its degradation states)

The main difference between these two measures is the design of the Markov model for different failure rates and mean down times. Depending on the type of failure-rates to take into consideration the states in the Markov model will differ both how they relate with each other and in amount. In order to make use of unavailability analysis the system has to be partitioned into repairable parts. The ASIC itself cannot be claimed repairable, but the ASIC together with its corresponding hardware platform may be replaced and hence is repairable by the means of the proof test interval limitation of its acceptable lifetime.

When handling multiple channelled systems the β–factor is used to model the probability of common cause failures to occur affecting both channels similarly.

The safety function is primarily embedded in an ASIC and is therefore not repairable according to the scope of this method, i.e. the system lifetime is limited of its proof test time interval.

Following parameters are necessary to determine before engaging the calculation of PFH or PFD,

-Proof test interval (T)
The proof test shall be periodically conducted in order to reveal any failures that have not been detected by the diagnostic tests of the system. When a failure is detected the system is completely restored. The proof test interval (T) determines the time (h) between proof tests performed.

-The mean time to restoration (MTTR)
The mean time to restoration is the average time for system recovery after the event of a detected failure. If the repair rate is assumed constant then: $\mu = 1/MTTR$ (all other types downtimes may be regarded in this parameter as well)

-The mean time to failure (MTTF)
The mean time to failure is the average time for a component/subsystem (independent function) or a system to fail in operation. Regarding individual components several parameters has to be considered. See [17] for further information. For subsystems and complete systems refer to section 8.6 for suitable methods. If the failure rate is assumed to be constant then: $\lambda = 1/MTTF$

-The safe failure fraction (SFF)
The safe failure fraction is the partition of safe failures out of all failures and is given by:

$$SFF = \frac{\sum \lambda_S + \sum \lambda_{DD}}{\sum \lambda_S + \sum \lambda_{DD} + \sum \lambda_{DU}}$$

-The diagnostic coverage (DC)
The diagnostic coverage is the partition of detected dangerous failures out of all dangerous failures and is given by:

$$DC = \frac{\sum \lambda_{DD}}{\sum \lambda_{DD} + \sum \lambda_{DU}}$$

The parameters SFF and DC are qualitative measures and it is recommended that these figures are based on a detailed FMEA.

The resulting parameters SFF and DC are used to point out which hardware safety integrity level is applicable (see Table 6 and Table 7). The PFD/PFH estimation is used to show conformance with the target hardware SIL by comparing the final PFD/PFH with Table 4 or Table 5)

8.7. Methods for validation of the design process and the verification process

Aim:
To verify that the required design and verification processes have been followed. In section 9.8 three methods are mentioned addressing the procedure of documentation system validation (DSV), the design process validation (DPV) and the design verification validation (DVV).

Performance:
The documentation system is verified by inspections and walkthroughs and reviews in order to show compliance with the requirements concerning documentation.

Applicability:
All design phases

Example:
Ensure that the verification has been followed according to the verification plan and that all required documents have been produced at the correct verification step.

Reference: [7]

8.7.1. Design review and inspections

Under following sub-sections methods for reviewing, verifying and validating the design and verification process carried out during the system design and verification.

8.7.1.1. Walkthroughs (system design review)

Aim:
To reveal discrepancies between the specification and the implementation and to describe the general functionality of the control system for a designer or an assessor.

Performance:
In walkthrough methods the designer(s) and assessor(s) manually go through the system design (process and verification) together and check the correctness and functionality. Usually some test cases, checklists and guidelines are used in this process. Specified functions of the safety-related system draft are examined and evaluated to ensure that the safety-related system complies with the requirements given in the specification. Doubts and potential weak points concerning the realization and use of the product are documented so that they may be resolved.
In contrast to an inspection, the author is active and the inspector is passive during the walk-through.

Result:
The result of the walkthrough procedure is a supplement and often an introductive input to most of the validation techniques and methods mentioned in this report. The result as such may not be sufficient to motivate fulfilment of specific requirements to be validated.

Applicability:
All phases in the design and safety validation process

References: [7]

8.7.1.2. Inspection (reviews and analysis)

Aim: To inspect that relevant requirements have been fulfilled.
Description: Specified functions of the safety-related system, the design documentation system and the design verification system are examined and evaluated to ensure that the safety-related system conforms to relevant requirements. The inspection shall be carried out using formalized and structured techniques such as checklists based on the overall safety requirements. Any deviations found shall be documented and resolved. In contrast to a walk-through, the author is passive and the inspector is active during the inspection procedure.

Result:
Documented fulfilment of relevant requirements.

Applicability:
All phases in the design and safety validation process

References: [7]

8.7.1.3. Fagan inspections

Aim: To reveal mistakes and faults in all phases of the HDL development.
Performance: A "formal" audit on quality assurance documents aimed at finding mistakes and faults. The inspection procedure consists of five stages: planning, preparation, inspection, rework and follow-up. Each of these stages has its own separate objective. The complete system design process (specification, design, coding and testing/verification) must be inspected. In the process the programmer reads the source code to a group who asks questions and analyses the program by using a checklist.

Result:
Shows conformity to the requirements for the HDL design, implementation and verification

Applicability:
All phases in the system verification and system validation process

References: [7]

8.7.1.4. Checklists

Aim:
To use a formal and structured approach for the system safety validation the design process and system verification process.

Performance:
A worksheet is established containing references to relevant requirements. Each requirement is accompanied with a set of concise questions that forms the criteria for success for that particular requirement. Checklists are also useful for the designer during the design and verification process in order to make sure that no relevant requirement is disregarded.

Result:
All results in the checklists are well justified by comments/remarks and references to other documentation. The final judgment of each requirement should be stated as a pass or fail conclusion. The use of such checklists simplifies the final conclusion on overall conformity. The checklist often forms the basis for inspections.

Applicability:
All phases in the design and safety validation process

Reference: [7]

8.7.2. Documentation system

In the following sub-sections methods for reviewing, verifying and validating a documentation system are listed.

8.7.2.1. Examination of documentation completeness

Aim: To check the completeness of the documentation required (from the design and verification process and from the safety validation process).

Performance: The required documentation is checked for completeness. A recommendation is to establish a checklist prior of commencing the design, verification and validation. The documentation system is then developed in accordance with the checklist through all phases.
Examples of useful data in such a checklist are:

- Document description
- Documentation number (reference/version/revision)
- Performer(s) (e.g. designer/assessor)
- Phase and date of creation (e.g. safety validation plan)
- Present document status
- Phase and date of finalization
- Approved by the responsible for the design phase, verification phase or validation phase (Pass/Fail)

Result: Declaration of conformity to relevant requirements for documentation.

Applicability: All phases in the design and safety validation process

Reference: None

8.7.2.2. Examination of the system specification

Aim: To examine the fulfilment of the requirements regarding design process specifications, design verification specifications and safety validation specifications and to make sure that all relevant requirements and specified functional properties and constraints have been considered.

Description: Various qualities of a specification document are assessed by an independent team. The assessment is preferably carried out through an examination performed by the designer team in conjunction with an independent team. After the examination, the independent team should be able to reconstruct the operational function of the system in an indisputable manner without referring to any further specifications.

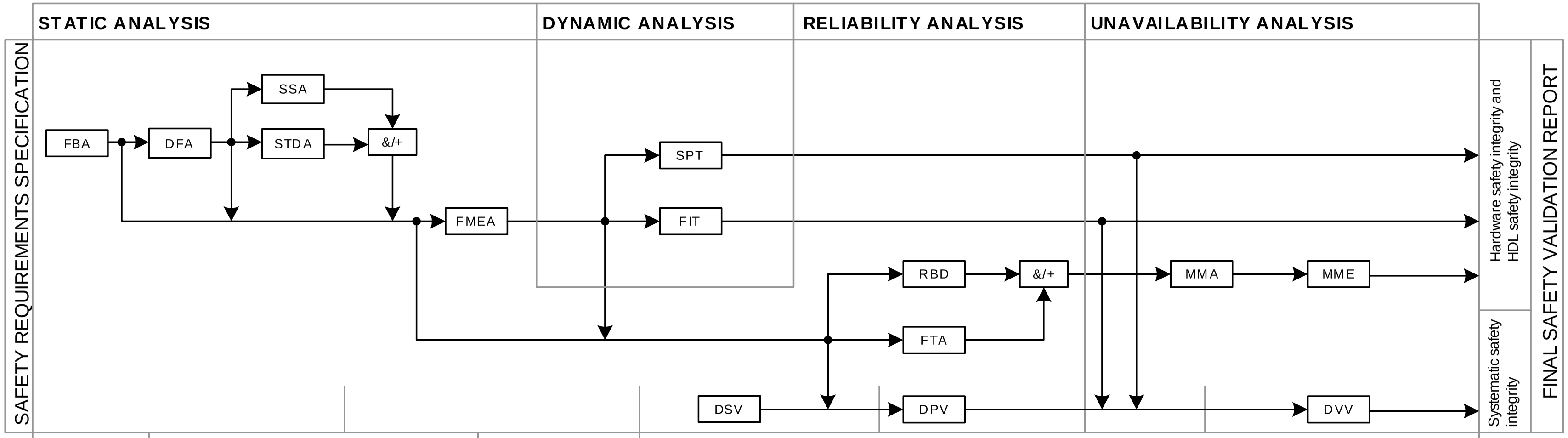
Result: Statement of conformity of relevant requirements for the system specifications and the adequateness of the system specifications

Applicability: All phases in the system verification and system validation process

Reference: [7], clause B.2.6

8.8. E/E/PES Hardware Safety Validation performance

SAFETY VALIDATION ROADMAP



STATIC ANALYSIS
DYNAMIC ANALYSIS
RELIABILITY ANALYSIS
UNAVAILABILITY ANALYSIS
SAFETY REQUIREMENTS SPECIFICATION
FBA
DFA
SSA
STDA
&/+
FMEA
SPT
FIT
RBD
&/+
FTA
MMA
MME
DSV
DPV
DVV
Hardware safety integrity and HDL safety integrity
Systematic safety integrity
FINAL SAFETY VALIDATION REPORT

Definition phase | Architectural design | Detailed design | Integration/implementation

Overview of the Validation process of hardware safety integrity and HDL safety intgrity

Provide a major understanding of:
-Overall design and functionality
-Safety-related parts and properties

Related requirements:
-Architectural constraints
-Level of fault tolerance
-Intended system behaviour at fault

Provide following information:
-Dangerous failures, component failure rates
-Undetected/detected failures
-Diagnostic test functional properties

Related requirements:
-SFF
-DC

Provide information that:
-Completes the FMEA
-Verifies previous analysis results
-Validates functional properties

Related reuqirements:
-Integration/functional tests
-Fault tolerance

Provide information about:
-Failure rates (for functional failures)
-Unintentional functional dependencies

Related requirements:
λ_D λ_{DD} λ_{DU} λ_{SD} β

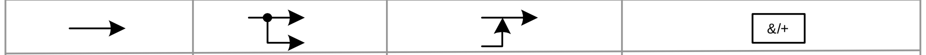
Provide information about:
-System failure operational states
-Unintentional failure operational states

Related requirements:
-T (Proof test inteval)
-PFD, PFH

Abbreviations:		Reference:
FBA	- Functional Block Analysis	Section 9.4.1
DFA	- Data Flow Analysis	Section 9.4.2
SSA	- Signal Sequence Analysis	Section 9.4.3
STDA	- State Transition Diagram Analysis	Section 9.4.4
FMEA	- Failure Mode and Effects Analysis	Section 9.4.5
SPT	- System Properties Testing (comprising functional and environmental testing techniques, used specifically for testing safety related properties)	Section 9.5.1-2
FIT	- Fault Injection Testing (comprising all previous mentioned techniqes, used specifically for testing safety related properties)	Section 9.5.3
RBD	- Reliability Block Diagram Analysis	Section 9.6.1
FTA	- Fault Tree Analysis	Section 9.6.2
MMA	- Markov Model Analysis	Section 9.6.3
MME	- Markov Model Evaluation	Section 9.6.4
DPV	- Design Process Validation	Section 9.7.1
DVV	- Design Verification Validation	Section 9.7.1
DSV	- Documentation System Validation	Section 9.7.2

Flow-chart description:
The flow chart above describes a possible safety-validation process. This flow-chart is intended to show how the different analysis techniqes mentioned in this report interact with each other during the complete validation process from the initial design review to the finishing functional tests and availability estimation. The assessor/designer may chose different techniques if suitable. The flow-chart begins at the left side with the source: Safety requirements specification and follows a roadmap of analysis techniqes to the final: Safety validation report on the right side. The analysis techniques are divided into four major areas: Static-, dynamic-, realibility- and unavailability analysis as shown on the top of the flow-chart. Below the flow-chart is a row with different design phases. This row is intended to show approximately where in the design process different techniques should be engaged. The row below the design phase row gives a general description of the most important (but not all) requirements to verify with these techniques. These descriptions also sub-divides the flow-chart in areas containing techniques by the short lines in the bottom of the flow-chart in order to show which techniques are most suitable to use for different types of requirements. Whenever a deviation is discovered that motivates a design modification, the requirements in section 9.1 applies. The modification will also impact on previous analysis results. Therefore the assessor/designer got to iterate back to an appropriate stage in the flow-chart. No arrows are added in the flow-chart for illustrating such iterations. Its up to the assessor/designer to motivate which results are affected by the modification. Each arrow in the flow-chart represents a document describing the performance and result of the corresponding analysis technique. All these documentations shall be included in the final safety validation report when finished.

Symbol definitions:



Source generates document described in the text above	The same result is used for two different analysis techniques	Different results are used as input to the same analysis technique	Either one out of several results or all results are used to the same analysis techique

9. Conclusion

The major conclusion of this work is the possibility of combining the means of functional safety with the very detailed design flow in the ECSS standard for ASIC/FPGA development. This is achieved by dividing the relevant part of the product development flow into following:

-Definition phase
-Architectural design phase
-Detailed design phase
-Design implementation phase
-Design verification phase
-Design validation phase

During all of these phases the designer/assessor has to consider the requirements that are not covered in this report and relate to the overall life cycle concept of IEC 61508. Every design decision or validation conclusion has to be related to the EUC and its corresponding risks. It is therefore not possible to determine the conformance to a SIL for a safety function only by the methods listed in this report. This report may only provide information that supports the determination of:

-the hardware safety integrity level;
-the systematic safety integrity level
-the HDL safety integrity level;

Sufficient information is compulsory before engaging the analysis and design according to the requirements listed in this report and in the target standards as mentioned above. Examples of checklists that may be used for guiding the application of this technical report is provided in appendix D.

According to the standard IEC 61508 safety-related functions have to depend both on the ASIC internal function and on the functionality of the hardware platform in which the ASIC is implemented; in order to prove compliance, the target system has to be designed with this prerequisite. If the designer implements all safety-related functions in an ASIC this report may also be used but no compliance to the standards may be proven.

Appendix A. Introducing reliability theory

A.1 Basic reliability measures in E/E/PE safety-related systems

This clause gives an introduction to the principles of reliability from which several of the analysis methods mentioned in this report are based. This overview is neither exhaustive nor complete in the subject. See [11] and other literature about fundamental probability theory for further or more detailed information.

R(t) denotes the reliability of a component at a given time t
Q(t) denotes the unreliability of a component at a given time t

Relation between the reliability function and the unreliability function:

$$Q(t) + R(t) = 1$$

P_r denotes the probability of correct operation (reliability) for a component, $P_r = R(a)$
where a = specific time

In a system where one of the events (functional failures) A or B occurs or both events occur causing a total system failure, the reliability R(t) is given by (intersection):

$$P_r = P(A \cap B) = P(A)P(B)$$

In a system where both the events (functional failures) A and B have to occur in order to cause a total system failure, the reliability R(t) is given by (union):

$$P_r = P(A \cup B) = P(A) + P(B) - P(A)P(B)$$

The intersection expression and the union expression only apply if the events (functional failures) A and B are completely independent of each other.

A.2 Probability density and distribution functions

pdf – probability density function (*f(x)*)
cdf – cumulative density function (*F(x)*)

Let *X* be a random continuous variable, then

$$P(a \le X < b) = \int_a^b f(x)dx$$

gives the probability of X to take a value in the given interval [a,b]. The pdf represents the relative frequency of failure multiplied by a function of time. If a = 0 and b = ∞ then the intersection expression always equals 1. The relation between the cdf and the pdf is provided by:

$$f(x) = \frac{d}{dx}F(x)$$

f(t) is normally expressed as the *normal distribution*. The random variable *X* is exchanged to *t* (time) since E/E/PE systems are time invariant.

A.3 The Reliability Function

At a given time *t* the reliability function is given by cdf as:

$$R(t) = 1 - F(t) = \int_{t}^{\infty} f(t)dt$$

This expression may be used for estimating the probability of a unit functioning at a time t.

A.4 The Failure Rate function

The failure rate function is a measure to find out the number of failures occurring per unit time and is mathematically defined as:

$$\lambda(t) = \frac{f(t)}{R(t)}$$

A.5 The Mean Life (MTTF)

The MTTF (Mean Time To Failure) is determined by the mean time of operation for a product and is calculated as:

$$MTTF = \int_{0}^{\infty} t \cdot f(t)dt$$

It's not recommended to consider MTTF solely as a reliability metric for the whole product. The distribution function may differ a lot although the mean life time is the same for different distribution functions. The MTTF metric shall not be confused with the MTBF (Mean Time Between Failure) metric. These two metrics are only identical if the failure rate is constant. The usage of MTBF only becomes meaningful when considering repairable systems.

There are also other statistical metrics (median life and modal life). For more information see [11].

A.6 Distributions

Some commonly used normal distributions for reliability are the following:

a) The normal distribution or Gaussian distribution:

$$f(t) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{t-\mu}{\sigma})^2}$$

b) The Weibull distribution:

$$f(t) = \frac{\beta}{\eta}(\frac{t-\gamma}{\eta})^{\beta-1} e^{-(\frac{t-\gamma}{\eta})^{\beta}}$$

This distribution is used when the failure rate is not constant during the component life time.

c) The exponential distribution:

$$f(t) = \lambda e^{-\lambda t}$$

This distribution is the most commonly used in reliability prediction. Using the previously mentioned equations it can be shown that,

The reliability function $R(t) = e^{-\lambda t}$
The failure rate $\lambda(t) = \lambda$ (constant)
The mean life $MTTF = \frac{1}{\lambda}$
The exponential distribution is also assumed to be used in this report. Given any pdf these metrics may be calculated using the equations shown in the previous subsections.

Appendix B. VHDL example

This section describes an exemplifying design where a large part of the functionality is implemented in one ASIC. The system is a double channelled and fail-safe and could be used in different applications such as a two-hand controller or a hold-to-run controller. The system is only an example used for clarification purpose and is not intended for realization. No requirements may be claimed as fulfilled when using the system in figure 1 unless a proper safety validation of the safety integrity is carried out according to relevant standards.

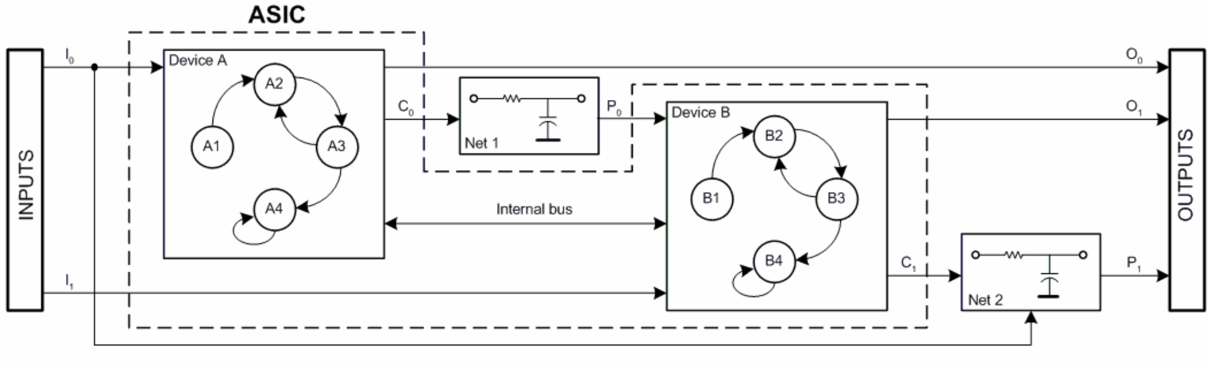


Figure B.1: VHDL code example system

B.1 System general description

The main purpose of this system is to keep a defined value on the outputs (O1 and O2) if the input values differ from the functional specification and if a fault occurs in any of the modules that forces the system into an operational mode that is not specified. The example is intended to describe the methods listed in this report. If connected to sensors and actuators such a system could be part of a safety function (if correctly designed and validated). By analyzing figure B.1 and gathering parts that affect the outputs, the system architecture may be derived as an FBD.

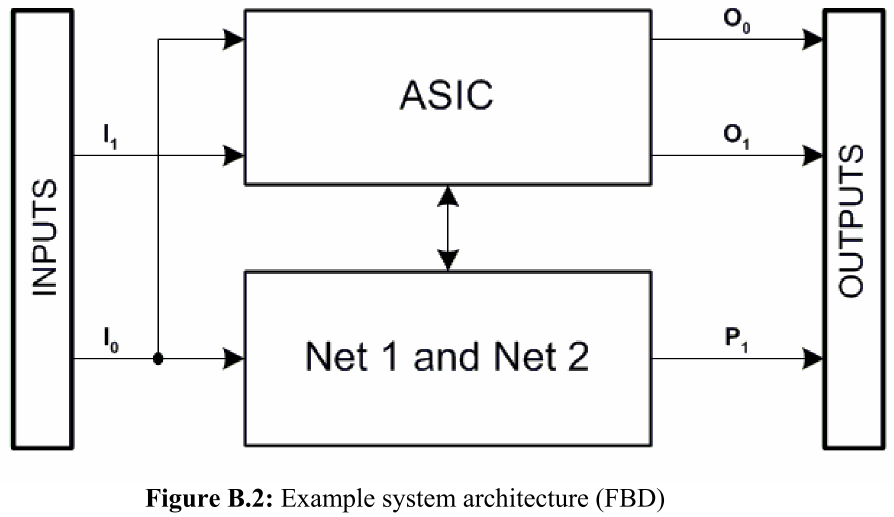


Figure B.2: Example system architecture (FBD)

Figure B.2 clearly shows that the system consists of two separated (redundant) channels that may independently affect the state of the outputs. It can also be seen in figure B.2 that both channels by some means have the possibility of monitoring each other. This architecture is therefore a 1oo2D-architecture according to IEC 61508 as the system may continue to fulfil the specified requirements although one channel has completely failed. “D” means that there is self-diagnosing function implemented in the system.

B.2 System detailed description

- ASIC

The ASIC functionality is divided into two devices (A and B) inside the ASIC. These in-circuit devices are physically well-separated and isolated from each other on the chip. The devices may communicate asynchronously with each other via an internal bus (which also may include other control signals).

- Passive nets

Both devices are connected to external filters (Net 1 and Net 2). Net 1 is used to supply power to device 2. Net 2 is used to supply power and hence to enable the OUTPUT interface. Net 2 depends on the ASIC and on external signals.

- Interfaces

The output interface contains electronics for signal level adaptation to the EUC. The output signal vectors O_0 and O_1 contain control signals (both for steering and for feed-back). The input interface (INPUTS) does also contain electronics for signal adaptation from the EUC. The signal vectors I_0 and I_1 contain sensor inputs, feed-back signals and two separated clock-sources, one for each device.

Table B.1: System overview

Modules	**Description**
INPUTS	Input signal adaptation, passive
OUTPUTS	Output signal adaptation, active – controlled by Net 2
Device A, B	Separated modules with control logic
Net 1, 2	Passive filters with suitable configurations and breaking frequencies
Signals	
I_0, I_1	Input vectors providing the ASIC with control signals, feed-back signals and two different clock sources running with different frequency
C_0, C_1	Control signals to the charge pumps dependent on the correct functionality in the devices (A and B)
P_0, P_1	Voltages dependent on the correct functionality of the devices (A and B)
Internal bus	Signal vector
O_0, O_1	Output control signal vectors entirely dependent on the input vectors and the correct functionality of device A and B

B.3 Short VHDL code description/introduction

Table B.1 describes the general functionality of device A and device B. The VHDL code is not complete and no detailed technical descriptions of solutions are added. The VHDL-code illustrates one approach as follows:

Library – Contains primitives and types (compare with include-files in C) that the synthesis-tool recognizes
Entity – Defines a “black box” with all input and output signals
Architecture – Defines the inner functionality of a certain entity or “black box” (i.e. describes how the outputs relates to the inputs)
The first thing that is done in the Architecture part is to define inner signals that are used within the entity. In this program is created a signal-type to which only four different values A1 – A4 (or 00, 01, 10, 11) may be assigned. This signal is used to define the different states of the state-machine illustrated in figure B.1. Some other signals are also defined (“send” and “rec”).
Process – A process is equivalent to a digital circuitry. The process is only sensitive to the parameters mentioned in its sensitivity list; in this case the clock and the reset signal. The process outputs will not

change until one of the parameters in the sensitivity list changes regardless of other process inputs (used for allocating latches and/or flip-flops, i.e. creating synchronous logic).
The first process condition becomes true during reset and determines the initial reset condition, i.e. sets all signals to their default value (including the state-variable being initialized to the state A1). This condition depends only on the reset-signal.

The second process condition may only become true if following are fulfilled:

- an event occurs in the clock signal (positive ore negative edge)
- the clock signal becomes high

On each positive clock edge this part of the process will change the process outputs. In practice the complete process will be "translated" into a combinatorial circuitry where all outputs are connected to a D-flip-flop that is controlled by the clock and the reset signal; see figure A.

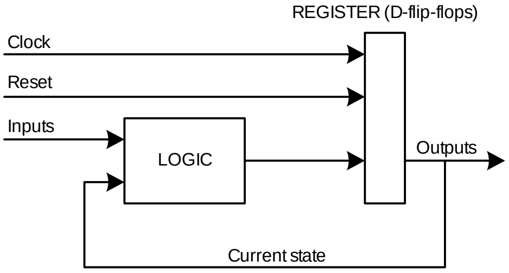


Figure B.3: The Ctrl_A process realization (STA).

The conditions in the synchronous part consist of a case-statement which is compared to the state-variable. All logic within one state will be processed in one clock cycle and all states are finalized by defining which state to enter at the next clock cycle. No control conditions are added in separate states in this example but some describing comments and the assignment of the next state accordingly to figure B.1. The state machine presently described is a Moore-state machine. In practice the states of such a state machine would most probably contain control signals to several sub-state machines that process the system inputs/outputs and would be huge in comparison to this example.

B.4 VHDL code functional description

Ctrl_A: process

A1: The state machine in this example begins with an initial state that is processed only once after a reset. This state performs all necessary initiations and reads a status value from a non-volatile memory indicating if the system wakes up from a previous safe-state operation. If any fault is detected when performing initial fault monitoring or if the status value of the current state is defined as state A4. If no fault is detected the next state shall be state A2.

A2: This state processes the system inputs and performs fault monitoring. A full UART buffer is indicated by the signal "rec". The handling of received messages should be performed using proper measures depending on the type of information. If a fault is detected (in the hardware or in the input signals or in the received message) the next state will be the safe-state (A4) or state A3.

A3: This state processes the system outputs and performs fault monitoring. When specific conditions are met a message may be composed, stored in the UART-buffer and sent by setting the signal "send" high. If a fault is detected (in the hardware or in the input signals or in the received message) the next state will be safe-state (A4) or state A2.

A4: Safe-state – writes the status to the non-volatile memory and forces all outputs to zero. The state-machine is locked at this state. If the power supply is switched off and on again, the system shall immediately be forced to this state.

If for some other reason another state than these four occurs (e.g. synthesis error so that the state signal consists of more than two wires) the state variable shall be forced to state A4 (safe-state).

UART
The UART is defined as a separate hardware device (process). No means for implementing UARTs are mentioned in this report. Such a process may require several sub-processes in order to function; this is not considered here.

Device B is implemented similarly as Device A.

B.5 Further aspects

For a designer several additional aspects has to be taken into consideration. Examples of questions that must have well-motivated answers are:

- what kind of communication topology to be used
- how to monitor the sequence in the state-machine
- which parameters shall affect the composition of the signals C0 and C1
- what type of signals shall be used as system outputs (dynamic or static)
- how to design the passive nets?
- what level of fault tolerance is acceptable before entering the safe-state

There is no perfect template for designing safety-related control systems because each application has its own properties and characteristics which affect the selection of the safety-principles to be applied. One general reflection is that fault handling measures always affect the performance of the system. The correct way to answer the questions listed above is to carry out the design process according to relevant standards or to follow the recommendations of this report.

Table B.2: VHDL code example

Device A

```
Library My_foundry_lib;
Use My_foundry_lib.std_logic_package.ALL;

Entity DeviceA is
Port(   I0_clk, D0_reset       : in    std_logic_package;
        I0_FB, I0_sensor       : in    std_logic_package;
        D0_bus_RX              : in    std_logic_package;
        D0_bus_TX              : out   std_logic_package;
        C0_out, O0_actuator : out   std_logic_package;
        D1_reset               : out   std_logic_package);
End entity DeviceA;

Architecture DeviceA_impl of DeviceA is
Type t_state is (A1, A2, A3, A4);
Signal state              : T_state;
Signal send, rec          : std_logic_package;
Begin
  Ctrl_A : Process(I0_clk, I0_reset)
  Begin
    If I0_reset = 1 then
        State            <=  A1;       --Reset settings
        O0_actuator      <= ‘ 0’;
        C0_out           <=  ‘0’;
        D1_reset         <=  ‘0’;
        Send             <= ‘0’;
    Elsif I0_clk’event and I0_clk = ‘1’ then
        Case state is
          When state = A1 =>
               -- Initiating sub-state machine that starts
               --  manipulating C0 and sends a reset-pulse
               -- (D1_reset) to device 2 when P0 becomes
               -- high enough. Perform system fault monitoring
               -- and check the non-volatile memory
               -- if waking up in failure mode of operation.
                 state <= A2; -- (if fault detected: state <= A4)
          When state = A2 =>
               -- Process inputs (I0_sensor, I0_FB), act on
               --  received messages (rec)
               -- and perform system fault monitoring and
               -- controlling. Maintain P0
                 state <= A3; -- (if fault detected: state <= A4)
          When state = A3 =>
               -- Process outputs (O0_actuator), compose and send
               -- messages (send) and perform system fault
               -- monitoring and controlling. Maintain P0
                 state <=  A2; -- (if fault detected: state <= A4)
          When state = A4 =>
               -- Safe-state – force all device outputs to zero
               -- Write fault-status into non-volatile memory
                 state <= A4;
          When others =>
                 state <= A4; -- Should not reach this state
        End case;
    End if;
  End process Ctrl_A;

  UART : process(RX, send, I0_clk)
  Begin
    -- Implementation of the UART to handle
    -- the internal bus. Full buffer indicated by rec.
    -- This process needs additional processes for the time
    -- reference.
  End UART;

End DeviceA_impl;
```

Device B

```
Library My_foundry_lib;
Use My_foundry_lib.std_logic_package.ALL;

Entity DeviceB is
Port(   I1_clk, D1_reset       : in      std_logic_package;
        I1_FB, I1_sensor       : in      std_logic_package;
        D1_bus_RX              : in      std_logic_package;
        D1_bus_TX              : out     std_logic_package;
        C1_out, O1_actuator : out     std_logic_package);
End entity DeviceB;


Architecture DeviceB_impl of DeviceB is
Type t_state is (B1, B2, B3, B4);
Signal state              : T_state;
Signal send, rec          : std_logic_package;
Begin
  Ctrl_B : Process(I1_clk, I1_reset)
  Begin
    If I0_reset = 1 then
        State            <=  B1;       --Reset settings
        O1_actuator      <=  ‘1’;
        C1_out           <=  ‘0’;
        Send             <=  ‘0’;

    Elsif I1_clk’event and I1_clk = ‘1’ then
        Case state is
          When state = B1 =>
               -- Initiating sub-state machine that
               -- start manipulating C1 until P1 sufficiently high
               -- for enabling the outputs (O0 and O1)
               -- Perform system fault monitoring and
               -- check the non-volatile memory if waking up in
               -- failure mode of operation.
                 state <= B2; -- (if fault detected: state <= B4)
          When state = B2 =>
               -- Process inputs (I1_sensor, I1_FB), act on
               --  received messages (rec)
               -- and perform system fault monitoring and
               -- controlling. Maintain P1
                 state <= B3; -- (if fault detected: state <= B4)
          When state = B3 =>
               -- Process outputs (O1_actuator), compose and send
               -- messages (send) and perform system fault
               -- monitoring and controlling. Maintain P1
                 state <=  B2; -- (if fault detected: state <= B4)
          When state = B4 =>
               -- Safe-state – force all device outputs to zero
               -- Write fault-status into non-volatile memory
                 state <= B4;
          When others =>
                 state <= B4; -- Should not reach this state
        End case;
    End if;
  End process Ctrl_B;

  UART : process(RX, send, I1_clk)
  Begin
    -- Implementation of the UART to handle
    -- the internal bus. Full buffer indicated by rec.
    -- This process needs additional processes for the time
    -- reference.
  End UART;

End DeviceB_impl;
```

Appendix C.1. Cross reference (IEC 61508-1)

Clause	Subclause	NT report reference
5.2	5.2.1	8.1
	5.2.2	8.1
	5.2.3	8.1
	5.2.4	8.1
	5.2.5	8.1
	5.2.6	8.1
	5.2.7	8.1
	5.2.8	8.1
	5.2.9	8.1
	5.2.10	8.1
	5.2.11	8.1
	5.2.12	8.1
6.2.	6.2.1	8.1
	6.2.2	8.1
	6.2.3	8.1
	6.2.4	8.1
	6.2.5	8.1
7.1.4	-	N/a
7.2.2	-	N/a
7.3.2	-	N/a
7.4.2	-	N/a
7.5.2	-	N/a
7.6.2	-	N/a
7.7.2	-	N/a
7.8.2	-	N/a
7.9.2	-	N/a
7.10.2	-	N/a
7.11	7.11.2	N/a
7.12	7.12.2	N/a
7.13	7.13.2	N/a
7.14	7.14.2	N/a
	7.14.3	N/a
	7.14.4	N/a
7.15	-	N/a
7.16	-	N/a
7.17	-	N/a
7.18	-	N/a
8.2	-	N/a

Appendix C.2. Cross reference (IEC 61508-2)

Clause	Subclause	NT report reference
7.2.2	7.2.2.1	N/a
	7.2.2.2	N/a
	7.2.2.3	N/a
7.2.3	7.2.3.1	N/a
	7.2.3.2	N/a
	7.2.3.3	N/a
7.3.2	7.3.2.1	8.2.2.3
	7.3.2.2	8.2.2.3
7.4.2	7.4.2.1	8.2.1
	7.4.2.2	8.2.1
	7.4.2.3	8.2.2.4
	7.4.2.4	N/a
	7.4.2.5	8.2.2.4
	7.4.2.6	N/a
	7.4.2.7	N/a
	7.4.2.8	N/a
	7.4.2.9	N/a
	7.4.2.10	N/a
	7.4.2.11	8.2.2.1
	7.4.2.12	8.2.2.1
	7.4.2.13	8.2.1.1
7.4.3	7.4.3.1.1	8.2.2.1
	7.4.3.1.2	8.2.2.1
	7.4.3.1.3	8.2.2.1
	7.4.3.1.4	8.2.2.1
	7.4.3.1.5	8.2.2.1
	7.4.3.1.6	8.2.2.1
	7.4.3.2.1	8.2.1.1
	7.4.3.2.2	8.2.2.2
	7.4.3.2.3	8.2.1.1
	7.4.3.2.4	8.2.1.1
	7.4.3.2.5	8.2.1.1
	7.4.3.2.6	N/a.
7.4.4	7.4.4.1	8.2.1.1
	7.4.4.2	8.2.1.1
	7.4.4.3	8.2.1.1
	7.4.4.4	8.2.1.1
	7.4.4.5	8.2.2.1
	7.4.4.6	N/a
7.4.5	7.4.5.1	8.2.1.1
	7.4.5.2	8.2.1.1
	7.4.5.3	8.2.1.1
7.4.6	7.4.6.1	8.2.1.1
	7.4.6.2	8.2.1.1
	7.4.6.3	8.2.1.1
7.4.7	7.4.7.1	8.2.5.2
	7.4.7.2	8.2.5.2
	7.4.7.3	8.2.5.2
	7.4.7.4	8.2.5.2
	7.4.7.5	8.2.5.2
	7.4.7.6	8.2.5.2
	7.4.7.7	N/a
	7.4.7.8	N/a
	7.4.7.9	N/a
	7.4.7.10	N/a
	7.4.7.11	N/a
	7.4.7.12	N/a
7.4.8	7.4.8.1	N/a
	7.4.8.2	N/a
7.5.2	7.5.2.1	8.2.3
	7.5.2.2	8.2.3
	7.5.2.3	8.2.3e
	7.5.2.4	8.2.3
	7.5.2.5	8.2.3
	7.5.2.6	8.2.3
	7.5.2.7	8.2.3

Clause	Subclause	NT report reference
7.6.2	7.6.2.1	N/a
	7.6.2.2	N/a
	7.6.2.3	N/a
	7.6.2.4	N/a
	7.6.2.5	N/a
7.7.2	7.7.2.1	9.3.1
	7.7.2.2	9.3.1
	7.7.2.3	9.3.1
	7.7.2.4	9.3.1
	7.7.2.5	9.3.1
	7.7.2.6	9.3.1
	7.7.2.7	9.3.1
7.8.2	7.8.2.1	9.3.2
	7.8.2.2	9.3.2
	7.8.2.3	9.3.2
	7.8.2.4	9.3.2
7.9.2	7.9.2.1	8.2.2.3
	7.9.2.2	8.2.2.3
	7.9.2.3	8.2.2.3
	7.9.2.4	8.2.2.3
	7.9.2.5	9.1
	7.9.2.6	9.1
	7.9.2.7	9.1
	7.9.2.8	9.1
	7.9.2.9	9.1
	7.9.2.10	9.1

Appendix C.3. Cross reference (IEC 61508-3)

Clause	Subclause	NT report reference
7.2.2	7.2.2.1	7.2.1
	7.2.2.2	7.2.1
	7.2.2.3	7.2.1
	7.2.2.4	7.2.1
	7.2.2.5	7.2.1
	7.2.2.6	7.2.1
	7.2.2.7	7.2.1
	7.2.2.8	7.2.1
	7.2.2.9	7.2.1
	7.2.2.10	7.2.1
	7.2.2.11	7.2.1
7.3.2	7.3.2.1	8.1
	7.3.2.2	8.1
	7.3.2.3	8.1
	7.3.2.4	8.1
	7.3.2.5	8.1
7.4.2	7.4.2.1	N/a
	7.4.2.2	7.2.2
	7.4.2.3	7.2.2
	7.4.2.4	7.2.2
	7.4.2.5	7.2.2
	7.4.2.6	7.2.2
	7.4.2.7	7.2.2
	7.4.2.8	N/a
	7.4.2.9	7.2.2.3
	7.4.2.10	7.2.2
	7.4.2.11	7.2.2
	7.4.2.12	7.2.2
7.4.3	7.4.3.1	N/a
	7.4.3.2	7.2.2
	7.4.3.3	7.2.2
7.4.4	7.4.4.1	N/a
	7.4.4.2	7.2.2
	7.4.4.3	7.2.2
	7.4.4.4	7.2.2
	7.4.4.5	7.2.2
	7.4.4.6	7.2.2
7.4.5	7.4.5.1	N/a
	7.4.5.2	7.2.2
	7.4.5.3	7.2.2
	7.4.5.4	7.2.2
	7.4.5.5	7.2.3
7.4.6	7.4.6.1	7.2.2
	7.4.6.2	7.2.2
7.4.7	7.4.7.1	8.1
	7.4.7.2	8.1
	7.4.7.3	8.1
	7.4.7.4	8.1
7.4.8	7.4.8.1	8.1
	7.4.8.2	8.1
	7.4.8.3	8.1
	7.4.8.4	8.1
	7.4.8.5	8.1
7.5.2	7.5.2.1-8	N/a
7.6.2	7.6.2	N/a
7.7.2	7.7.2.1-8	N/a
7.8.2	7.8.2.1-10	N/a

Clause	Subclause	NT report reference
7.9.2	7.9.2.1	8.2
	7.9.2.2	8.2
	7.9.2.3	8.1
	7.9.2.4	8.2
	7.9.2.5	8.2
	7.9.2.6	8.2
	7.9.2.7	8.3
	7.9.2.8	8.1
	7.9.2.9	8.2
	7.9.2.10	8.2
	7.9.2.11	8.2
	7.9.2.12	8.1
	7.9.2.13	N/a

Appendix D. Guiding checklists

Every row in the table below is to be considered as a requirement by the means of this report.

Table D.1 Initiating set of documentation, related to the EUC, required to be produced before the appliance of this report

Ref.	Description	Related section	Notes, remarks	Requirement fulfilled ? (Yes/No)	Referred document(s)
A.1.1	Is/are the document(s) describing the concept available?	8.1			
A.1.2	Is/are the document(s) describing the overall scope definition available?	8.1			
A.1.3	Is/are the document(s) describing the hazard and risk analysis available?	8.1			
A.1.4	Is/are the document(s) describing the overall safety requirements available?	8.1			
A.1.5	Is/are the document(s) describing the safety requirements allocation available?	8.1			
A.1.6	Is/are the document(s) safety requirements specification available (part of the functional specification)?	8.1			
A.1.7	Does/do the safety requirements specification document(s) include the safety function specification?	8.2			
A.1.8	Does/do the safety requirements specification document(s) include the safety integrity specification?	8.2			
A.1.9	Does the functional specification include an identification of the ASIC requirements specification?	8.2			
A.1.10	Does the functional specification include an ASIC feasibility study?	8.2			
A.1.11	Does the functional specification include a quality related ASIC risk analysis	8.2			
A.1.12	Does the functional specification include an ASIC development plan?	8.2			
A.1.13	Has an initial design review been performed and documented?	8.2			

Table D.2 General requirements (Definition phase)

Ref.	**Description**	**Related section**	**Notes, remarks**	**Requirement fulfilled ? (Yes/No)**	**Referred document(s)**
A.2.1	Is the control system to be designed according to the safety requirements specification?	8.2.1			
A.2.2	Does the safety requirements specification consider the diagnostic test interval to be achieved?	8.2.1.1 a)			
A.2.3	Does the safety requirements specification include requirements on the system behaviour on detection of a fault?	8.2.1.1 b)			
A.2.4	Does the safety requirements specification include requirements for controlling systematic failures?	8.2.1.1 c)			
A.2.5	Does the safety requirements specification regard human capabilities and limitations?	8.2.1.1 d)			
A.2.6	Are the principles mentioned in the report section 7.2 – 7.4 considered in order to meet the general requirements?	8.2.1.1 e)			
A.2.7	Are the general requirements for the design performance considered (techniques and measures, documentation and maintainability/testability)?	8.2.1.1 f-h)			
A.2.8	Is de-rating used (>0.67) as far as possible for all system components?	8.2.1.1 i)			
A.2.9	Are all mentioned general ASIC requirements considered?	8.2.1.2			
A.2.10	Does the definition phase result in a complete requirement specification?	-			

Table D.3 Requirements in the architectural design phase (to be included in the system design requirements specification and in the architectural design)

Ref.	Description	Related section	Notes, remarks	Requirement fulfilled ? (Yes/No)	Referred document(s)
A.3.1	Is the safety integrity level translated into: -HDL safety integrity level? -Hardware safety integrity level? -Systematic safety integrity level?	8.2.2			
A.3.2	Are the ASIC design and test bench design planned to be carried out in accordance with the design flow in figure 7	8.2.2			
A.3.3	Is the planned design based on a decomposition of subsystems/modules?	8.2.2.1			
A.3.4	Does every subsystem/module have a specified design?	8.2.2.1			
A.3.5	Does every subsystem/module have a specified set of integration tests?	8.2.2.1			
A.3.6	Is it possible to depict the overall architectural design as in figure 8?	8.2.2.1			
A.3.7	Is the design divided in type A and type B components, according to table 6 and 7?	8.2.2.1			
A.3.8	Are the SFF and level of hardware fault tolerance determined?	8.2.2.1			
A.3.9	Is table 8 integrated to its full extent in the system design specification?	8.2.2.2			
A.3.10	Has a verification and validation plan been established (according to all requirements in this section)?	8.2.2.3			
A.3.11	Has this verification and validation plan been established concurrently to the architectural design?	8.2.2.3			
A.3.12	Does the verification plan describe how to verify the ASIC in the subsequent design phases?	8.2.2.3			

Ref.	Description	Related section	Notes, remarks	Requirement fulfilled ? (Yes/No)	Referred document(s)
A.3.13	Are requirements on the HDL code quality included in the system design specification?	8.2.2.4			
A.3.14	Are requirements on the HDL code quality included in the test bench design specification?	8.2.2.4			
A.3.15	Has a system for version and revision and modification due to the design handling been established and included in the system design specification?	8.2.2.4			
A.3.16	Is a plan established for reviewing of the HDL code and the script files used?	8.2.2.4			
A.3.17	Has the architectural design phase resulted in the outputs specified?	8.2.2.4			
A.3.18	Have all requirements in the architectural phase been verified and approved in order to close this phase and open the detailed design phase?	-			

Table D.4 Detailed design phase

Ref.	Description	Related section	Notes, remarks	Requirement fulfilled? (Yes/No)	Referred document(s)
A.4.1	Is the modified V-model described in figure 11 in conjunction with the design flow in figure 8 used during the detailed design?	8.2.3			
A.4.2	Are all requirements a)-g) regarded?	8.2.3			
A.4.3	Have sufficient integration and verification tests been performed during the detailed design?	8.2.3			
A.4.4	Has the hardware platform PCB been designed according to the design specification and regarding the results of the verification tests performed?	8.2.3			
A.4.5	Is the ASIC layout designed in accordance with the design flow in figure 9?	8.2.4			
A.4.6	Is an initiating plan established for organizing the design on the chip (floor planning)?	8.2.4			
A.4.7	Has a sub-block place and route action been performed?	8.2.4			
A.4.8	Has the ASIC layout been verified by the means of a design rule check (DRC)?	8.2.4			
A.4.9	Has the ASIC layout been verified by the means of an equivalence check between the gate level netlist and the layout (LVS)?	8.2.4			
A.4.10	Has actual timing data being extracted (Timing extraction) due to the layout (actual wire-lengths present)?	8.2.4			
A.4.11	Is ATPG used in order to verify the different steps in the implementation?	8.2.4			
A.4.12	Have means for post production tests been established?	8.2.4			
A.4.13	Has the prototype ASIC been manufactured (by the means of a foundry process or in a fine-grained FPGA)?	8.2.5.1			
A.4.14	Is the system implemented in accordance with the system design specification?	8.2.5.1			

Ref.	Description	Related section	Notes, remarks	Requirement fulfilled ? (Yes/No)	Referred document(s)
A.4.15	Are all safety functions identified and documented?	8.2.5.2			
A.4.16	Is all required documentation for the subsystem available?	8.2.5.2 a)			
A.4.17	Has the rate of failure been determined according to the requirements?	8.2.5.2 b)			
A.4.18	Have the requirements for components that are considered as proven in use been fulfilled?	8.2.5.2 c)			
A.4.19	Have all requirements in the detailed design phase been verified and approved in order to close this phase and open the detailed design phase	-			

Table D.5 Verification process

Ref.	Description	Related section	Notes, remarks	Requirement fulfilled ? (Yes/No)	Referred document(s)
A.5.1	Has the ASIC and functional safety verification begun at an early stage in the detailed design phase?	9.1			
A.5.2	Is the design verification plan an outcome from the architectural design phase?	9.1			
A.5.3	Have the verification actions in the design flow viewed in figure 8 been followed?	9.1			
A.5.4	Has the modified V-model in figure 11 been used during the detailed design?	9.1			
A.5.5	Have the methods a)-c) for static property check been used during the verification steps in the detailed design?	9.1.1			
A.5.6	Have the methods a)-g) for fault injection and simulation been used during the verification steps in the detailed design?	9.1.2			
A.5.7	Have the used test benches been verified as the actual design?	9.1.3			
A.5.8	Has all the required verification documentation from the architectural design been produced?	9.2.1			

Ref.	Description	Related section	Notes, remarks	Requirement fulfilled ? (Yes/No)	Referred document(s)
A.5.9	Has all the required verification documentation from the detailed design phase been produced?	9.2.2			
A.5.10	Has all the required documentation form the layout design phase been produced?	9.2.3			
A.5.11	Have all requirements in the design verification process been verified and approved in order to close this phase and open the safety validation process?	-			

Table D.6 Safety validation process

Ref.	Description	Related section	Notes, remarks	Requirement fulfilled ? (Yes/No)	Referred document(s)
A.6.1	Has the E/E/PE safety requirements specification been composed according to the requirements?	9.3			
A.6.2	Are all requirements on safety validation regarded when starting the safety validation?	9.3.1			
A.6.3	Has an action plan been established according to the requirements that handle modifications of the system due to the validation results?	9.3.2			
A.6.4	Have suitable static analysis methods been used in order to show the design compliance with the requirements in the safety validation plan?	9.4			
A.6.5	Have suitable dynamic analysis methods been used in order to show compliance with the requirements in the validation plan?	9.5			
A.6.6	Have suitable analysis methods been applied in order to estimate the probability of failure and the system availability?	9.6			
A.6.7	Have suitable methods been used for verify and validate the design and verification process (including the documentation system)?	9.7			

Ref.	**Description**	**Related section**	**Notes, remarks**	**Requirement fulfilled ? (Yes/No)**	**Referred document(s)**
A.6.8	Has the system been verified and validated to correspond with the required SIL due to the: -Hardware safety integrity -HDL safety integrity -Systematic safety integrity And hence providing sufficient risk reduction?	9.8			
A.6.9	Have all requirements not regarded by this report been fulfilled in order to prove conformity to the standards [1] and [2] - [4] ?	10			
A.6.10	Have all requirements in the design safety validation process been verified and approved in order to close this phase and release the final system to the end-user?	-			

norden

Nordic Innovation Centre

Return address:
Nordic Innovation Centre,
Holbergs gate 1,
NO-0166 Oslo, Norway

NORDTEST

NORDTEST is a Nordic Innovation Centre brand offering competence and expertice in the field of harmonizing of norms and methods,a large Nordic net-work of experts, more than 650 recommended Nordic testing methods and 550 published technical reports.

www.nordicinnovation.net

Nordic Innovation Centre

The Nordic Innovation Centre initiates and finances activities that enhance innovation collaboration and develop and maintain a smoothly functioning market in the Nordic region.

The Centre works primarily with small and medium-sized companies (SMEs) in the Nordic countries. Other important partners are those most closely involved with innovation and market surveillance, such as industrial organisations and interest groups, research institutions and public authorities.

The Nordic Innovation Centre is an institution under the Nordic Council of Ministers. Its secretariat is in Oslo.

For more information: www.nordicinnovation.net