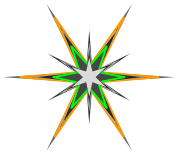


S T S A R C E S

Standards for Safety Related Complex Electronic Systems

Annex 10

Safety Validation of Complex Components

Validation Tests

Final Report of WP3.3

Klaus Bosch & Frank Mayer

TÜV PRODUCT SERVICE GMBH

European Project STSARCES

Contract SMT 4CT97-2191

-

Abstract

This paper sums up the results of the research on Work Package 3.3 “Safety Validation of Complex Components – Validation Tests”. The objective of this work package was to collect state of the art validation test methods and to assess the effectiveness of these test methods in the special context of complex components. Suitable sets of test methods will be recommended for the different types of complex components and these sets will be assigned to the safety categories of EN 954-1.

This paper is structured in two main parts, as follows: First, the results of the work on WP 3.3, as the main part of this contribution. These results and conclusions are presented as compressed and as short as possible, to allow a more easy integration of the main points into the final report for the overall STSARCES project. Second, a number of appendices, that give the required background and in-depth information on the topics that are addressed in the first part. Although called “appendix”, this second part contains vulnerable working results of WP 3.3 and is intended to help for a thorough understanding of the first part of this final report.

-

1 Table Of Contents

-

-

-

-

-

2 Preface

The STSARCES – Standards for Safety Related Complex Electronic Systems – project is funded by the European Commission SMT programme. Main objective of the STSARCES project is to render the machinery necessary for European industry as safe as possible from the design stage onwards. The STSARCES project is divided into several research work packages. This paper sums up the results of the research on Work Package 3.3 “Safety Validation of Complex Components – Validation Tests”.

This report was prepared by Dipl.-Ing. Klaus Bosch, TÜV Product Service GmbH, and Dipl.-Inf. Frank Mayer, Fraunhofer Institut für Integrierte Schaltungen.

-

3 Overview

This overview gives a short glance at the objectives of the WP 3.3 and how to proceed.

3.1 What are the objectives of WP 3.3 ?

Nowadays, complex components, like microprocessors, memories (RAM, EPROM, Flash), programmable logic (PLD, FPGA), ASICs and other high integrated circuits may be used as building blocks for safety related electronics. Due to large scale integration, it is possible today to integrate a whole system – that required a board or a assembly of boards some years ago – onto a single chip.

In the context of the DIN V 0801, EN 954 and IEC 61508, different validation tests are well known and already described in those released or draft standards. But, this type of validation tests might fail short when confronted with complexities of several thousands – or up to millions – of interacting logic primitives and memory cells.

Thus, the objectives of WP 3.3 is to fill this gap between existing validation tests and the requirements for a trustworthy safety validation of a single complex component or a system build of several complex components. In the reminder of this text, the terms “complex component” and “complex system” are used interchangeable; as it is shown in more detail in chapter C.3 both may be only different representations of the same functionality. A complex “system” that required a number of boards some time ago may be implemented in a single “component” today.

3.2 How to proceed ?

To get a standardised package of validation tests for complex components, it is necessary to go ahead step by step.

The **first step** is to consider all state of the art validation tests which are used up to now for complex or semi-complex components. These methods were evaluated and assessed.

The **second step** is to consider the changes in production and design of very complex components. Very complex components are designed with mighty software tools and special software languages (e. g. VHDL). Therefore, verification and

-

validation steps based on the different design flows were described and possible hazards were identified.

The **third step** is to find out suitable sets of validation tests for complex components. It was required to define a new approach for verification and validation of complex components.

-

4 State of the Art Validation Test Methods

4.1 Safety Validation Concepts

Safety validation nowadays is described in a couple of international standards. The most important of these standards are IEC 61508, part. 1 to 7, DIN V VDE 0801 and appendix A1, EN 50128 and EN 50129 (the last two especially for railway applications of programmable electronic systems)

All these standards are defining methods of safety validation and methods of planning the safety validation. Especially in the IEC 61508 one of the main topics is the planning of the safety validation by using e. g. V&V-plan (verification & validation plans).

4.2 Validation Test Methods

The following text summarises and comments the state of the art validations test methods. In the following table „Safety validation tests for electronic systems“ the tests are assigned to the safety categories (CAT 1 - 4) introduced in EN 954-1.

The following notation is used in Table 1 for each method. A qualitative rating (“high” – “medium” – “low”) for the required test coverage is given; this may be translated into more measurable figures (quantitative rating) by using the definitions in the IEC 61508.

qualitative rating for this method (first line)	HR	method is highly recommended for this safety category
	R	method is recommended for this safety category
	–	method is not required, but may be used
required test coverage of this method (second line)	high[1]	a high degree test coverage is required
	medium	a medium degree of test coverage is required
	low	a acceptable degree of test coverage is required

[1] “high” replaces the misleading “mandatory” used in tables in existing standards, e. g. in the 61508.

-

Technique/measure	Cat 1,2	Cat 3	Cat 4
Functional testing	HR high	HR high	HR high
Functional testing under environmental conditions	HR high	HR high	HR high
Interference immunity testing	HR high	HR high	HR high
Fault injection testing	HR high	HR high	HR high
Expanded functional testing	– low	HR low	HR high
Surge immunity testing	– low	– low	– medium
Black box testing	R low	R low	R medium
Statistical testing	– low	– low	R medium
“Worst case” testing	– low	– low	R medium

Table 1: Safety validation tests for electronic systems

As listed above, a couple of validation test methods are already described in released and draft standards. Additional details for each method, based on the descriptions of the IEC 61508, and comments on the usability in the context of “complex components” may be found in Appendix A: “Safety Validation Methods”.

The detailed analysis of these existing methods reveals a number of potential limitations when confronted with the validation of a complex component:

- ***complexity:*** the component might be far to complex for an adequate validation; it is not possible to reach the coverage figures from Table 1 for the given category.
- ***controllability:*** interconnections and logic inside the component is not directly controllable.
- ***observability:*** the reaction to input stimuli might not be observable; attaching probes is either not possible (internal signals) or affects the test results.

-

Moreover, an additional drawback of the listed validation tests is the fact that they are applicable only very late in the development process, because a “real” hardware is required to run most of the tests. The system that is used during the validation test has to be as close as possible to the one that will be used in the field, otherwise the result of the validation test is not expressive at all.

Using validation testing late in the development process incorporates the risk that every hazard found during the test is likely to result in a time consuming re-design and product improvement process. Because potential problems might be found very late in the product development process, the overall development effort and time to market may be very hard to estimate in advance.

4.3 Conclusion

For complex components, validation testing has to go “beyond the surface” of the component and is advised much earlier in the development process. For example, functional testing has to start at module level – using modules with very limited complexity – and has to accompany the hierarchical (bottom up) integration of the modules to more complex building blocks, step by step, until the complete functionality of a “complex component” is reached and all application and safety requirements are met.

To classify this proposed validation test scheme, it is useful to give a short reminder on the general definitions (ISO 8402) for *validation* and *verification* first:

Validation := „Confirmation by examination and provision of objective evidence that the particular requirements for a specific intended use are fulfilled.“

Validation is the activity of demonstrating that the safety-related system under consideration, before or after installation, meets in all respects the safety requirements specification for that safety-related system. Therefore, for example, software validation means confirming by examination and provision of objective evidence that the software satisfies the software safety requirements specification.

Verification := „Confirmation by examination and provision of objective evidence that the specific requirements have been fulfilled.“

Verification activities include:

- reviews on outputs (documents from all phases of the safety lifecycle) to ensure compliance with the objectives and requirements of the phase, taking into account the specific inputs to that phase;

-

- design reviews;
- tests performed on the designed products to ensure that they perform according to their specification;
- integration tests performed where different parts of a system are put together in a step by step manner and by the performance of environmental tests to ensure that all the parts work together in the specified manner.

In the context of these definitions, our proposed *validation* test scheme results in the *sum of independent verification* steps during the implementations process. The complete, uninterrupted sequence of verification steps provides the objective evidence („validation“) that the final result (e. g. the programmed FPGA) fulfils the initial requirements for the intended use and the required safety category.

-

5 Component Design and Production

5.1 State of the Art Design Process

Prior to define adequate validation tests – or, as concluded in the previous chapter: a continues, uninterrupted chain of verification steps parallel to the design process – we have to focus on state of the art PLD, FPGA and ASIC design process.

5.1.1 Technology

The term “complex component” may be applied to a wide variety of devices. The range spans different process technologies, different design and implementation methodologies as well as different levels of complexity. To clarify the term “complex component” in the context of safety validation, some typical examples for different technologies are given in Appendix B: “Technology Overview”.

5.1.2 Complexity

In Appendix C: “Complexity Metrics”, an attempt is made to objectively “measure” the complexity of a component, based on different complexity metrics. This helps to judge the effectiveness of the different validation methods for different level of complexity of the device under test.

The metrics listed and described in Appendix C: “Complexity Metrics” are well known and some of them are referenced in other contributions to the STSARCES project. E.g. a component is considered to be complex if it has “more than 1000 gates and / or more than 24 pins”. The problem with all those metrics is the fact that no direct link from the measurable “complexity” to the required level of validation has been found up to now. This implies that it is not possible to categorise the required type or effectiveness of verification or validation tests based on any of the listed complexity metrics. Additional work and a different approach – presented as part of the chapter “Validation Tests for Complex Components” – was necessary to get this linkage between “complexity” and validation effort.

-

5.1.3 Design Flow

Appendix E: “PLD / FPGA Design Flow” and Appendix D: “ASIC Design Flow” shows the different methodologies, design steps and tools typically used for the development of complex components.

5.1.4 Conclusion

For safety-related integrated circuits, the different device types require different validation concepts. For example, the layout and placement of the cells of a gate array or a FPGA is fixed; components based on these predefined structures are manufactured in larger numbers, thus the structure itself might be considered as „proven in use“ after some time. For the various types of ASICs and standard ICs, the structure is defined during the layout process. Thus, especially for deep sub-micron processes, interference between neighbouring cells or interconnections are possible, with actual influence on the chips functionality. It is obvious that this situation has to be considered during validation testing and fault injection.

5.2 Linkage between the Design and Validation Process

5.2.1 Phase Model

It is useful to identify the major steps that lead to a production-ready component. This “phase model” is intended to be more general as the two design flows given above. Based on the phase model from the IEC 61508, the following phases are identified:

(1) ***Specification***: Textual or formal description of the device’s functionality

(2) ***Design Description***: Formal description (e. g. Boolean Equations, Schematic, (V)HDL) that may be automatically translated into a fusemap / bitstream (PLD, FPGA) or gate level netlist (Gate Array, ASIC).

(3) ***Implementation***: Transformation of the design description into a netlist / fusemap / bitstream that may be used to produce or program the component. This phase is subdivided into two phases: “Implementation I” maps the design description into the primitives of the target device (logic blocks, gates), “Implementation II” produces the final information required for the component production or programming (fusemap or bitstream file, layout database).

-

(4) ***Production***: Production (programming) of the component, based on the output of the implementation phase.

(5) ***Post Production***: The component is available for standard system integration and validation tests.

Phase	**Output (PLD / FPGA)**	**Output (Gate Array, ASIC)**	**level of detail**	**usability for formal or simulation based verification**
Specification	Specification Documents (pure textual or semi-formal, e. g. using block and state diagrams, pseudo-code)		"high level" description with low level of detail	partial (only for those parts described semi-formal)
Design Description	Formal description of the functionality of the device, usable for automatic translation.		(virtual) components, blocks, processes RTL ("register transfer level")	all[2] functional aspects (RTL level) no explicit information about timing behaviour
Implementation I	primitives netlist, (propriety) database	gate level netlist	FPGA primitives, ASIC gates; interconnections estimated timing behaviour Gate Level	all functional aspects (Gate Level) estimated timing behaviour
Implementation II	fusemap / bitstream	layout database (e. g. GDS-II)	physical placement and interconnection	all functional aspects (Gate Level) actual timing behaviour
Production	programmed device programmed configuration PROM	packaged and tested device	component	device characteristics (overall functionality, timing)
Post Production	Board / System		"black box"	black box testing only[3]

Table 2: Phase Model

[2] This is true if the functionality is independent of the timing behaviour, e. g. for a pure synchronous design that will be clocked with a frequency less than 1 / (maximum path delay).

[3] Although "on chip" measurements and tests are theoretical possible (e. g. E-Beam test); not feasible in most cases, because this would require high specialised equipment.

-

5.2.2 Validation Tests & Phase Model

With the information from Table 2 it is now possible to map the known validation tests (Table 1) to the phases of our model. This is detailed in Table 3.

Phase	Validation Tests								
	Functional Testing	Funct. Testing under environ. conditions	Interference Immunity Testing	Fault Injection Testing	Expanded Functional Testing	Surge Immunity Testing	Black Box Testing	Statistical Testing	Worst Case Testing
Specification	Note (1)								
Design Description	Note (2)			Note (6)	Note (2)			Note (2)	
Implementation I	Note (2)	Note (4)		Note (6)	Note (2)			Note (2)	Note (4)
Implementation II	Note (2)	Note (4)	Note (5)	Note (6)					Note (4)
Production				Note (7)					
Post Production	Note (3)	Note (3)	Note (5)	Note (3)	Note (3)	Note (5)	Note (8)	Note (3)	Note (3)

Table 3: Validation Testing linked to Phase Model

The following notation is used in Table 3 for each method:

rating for applicability of this method in this phase

- test is not useable or expressive in this phase
- test might be used in this phase (with limitation, see Notes)
- test is well suited for this phase

Notes:

(1) Functional Testing in Specification Phase: Only if semi-formal methods are used during specification. Results are valid only if the implementation is derived directly from the specification and this may be verified.

(2) Functional, Expanded Functional and Statistical Testing in Design Description and Implementation Phase: Depending on design description methodology. For pure synchronous designs, functional testing in the design description phase might be adequate. Timing-related functionality aspects need to be addressed in the Implementation Phase.

(3) Validation Tests in the Post Production Phase (of the component itself): In the Post Production Phase, two different aspects need to be distinguished: validation tests that concentrate on the

-

component itself and validation / integration tests for the board or system this component is used in. Table 3 refers to the component itself, thus the applicability of the validation tests is limited in most cases (for details, see chapter “State of the Art Validation Test Methods”). Nevertheless, through integration and validation testing at board / system level is advised, as already described in existing standards.

(4) Testing under Environmental or Worst Case Conditions: This refers to the typical environmental condition that are considered for integrated circuits: Temperature, Supply Voltage and Process Deviation. Timing information – for path delays, setup- and hold times – that may be used for formal or simulation based validation testing is available for “best”, “typical” and “worst” case environmental conditions (see Table 4 for details).

timing condition	temperature	supply voltage	process deviation	Remark
“best”	lowest specified	highest specified	best (fastest) process	best case for path delay, but worst case for required setup and hold times
“typical”	typical (e. g. 25℃)	nominal	typical	typical case (not meaningful in most cases)
“worst”	highest specified (on chip)	lowest specified	worst (slowest) process	worst case for path delay (determines max. clock frequency for synchronous design)

Table 4: Definition of “best”, “typical” and “worst” operating conditions

(5) Interference Immunity, Surge Immunity Testing: The behaviour of a component during surge immunity testing is dependent on various parameters; not all of them may be quantified during the implementation phase, nor is it possible to rely on existing models for a precise estimation. Thus, lump estimation and testing is possible without the final component.

(6) Fault Injection Testing: This may be done with different levels of detail, e. g. looking a functional aspects during the design description phase and at stuck-at and coupling faults in the implementation phase.

(7) Production Test (Gate Array, ASIC only): It is important to clearly distinguish fault injection testing during the design process and the production test for Gate Arrays and ASIC. Both methods use the same fault models (e. g. “single-stuck-at”), but for different types of analysis; thus it is not possible to mix the results of the two methods (e. g. to apply the fault coverage figure for the production test to fault injection testing in the design process).

(8) Black Box Testing: Treating the complex component itself as “black box”.

5.2.3 Completeness

Moving validation tests to an earlier phase in the design and implementation process has the potential weakness that the result of a test carried out in an early design phase might be invalidated during the subsequent implementation steps. Thus it is

-

required to check the output of every implementation step against its input (= “verification”). This is shown in Figure 1 and results in additional verification tasks required in the validation process.

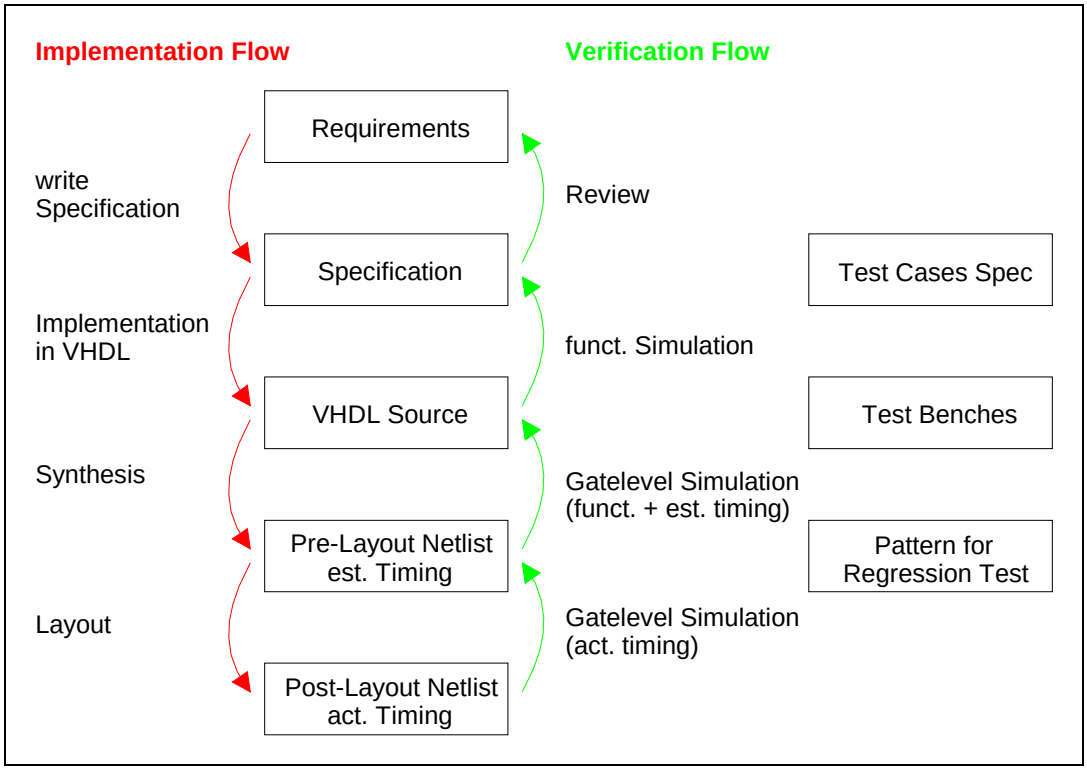


Figure 1: Implementation and Verification

The following table (Table 5) links the various work packages of the PLD/FPGA (Appendix E: ”PLD / FPGA Design Flow”) and ASIC (Appendix D: “ASIC Design Flow”) design flow to the phase model. Potential hazards – faults that may invalidate the result of a validation done earlier – are listed and possible countermeasures (verification concepts) are derived. A more detailed description of the work packages and more information on the potential hazards may be found in the two appendices.

Note: The first entry in the “Hazards” column for each Work Package is usually blank; the belonging entry in the “Verification done” column lists the standard verification tasks for this package.

-

Phase	Work Package in Design Flow	Hazards	Verification done
Specification	Textual Description		by internal and independent review
		no automated check possible	by review
	Specification, using semi-formal methods (state diagrams, flow charts, spreadsheets, block diagram		by internal and independent review by using the method itself, supported by automated tools by formal analysis and simulation of the specification
		same tool used for description and verification	by review later in design flow
		no automated check done	by review later in design flow
		partial verification, insufficient quality of the test cases	by review later in design flow
		no direct link to implementation (e. g. code generation)	by review later in design flow
	Modelling (behavioural model, written in behavioural VHDL or C code)		by internal and independent review by formal analysis or simulation of the model by using the model in the system context
		partial verification, insufficient quality of the test cases	by review later in design flow
		no direct link to the implementation (limited accuracy of the model	by review later in design flow

-

Design Description	Boolean Entry		by walk-trough (review) by functional simulation (if supported)
		error prone, low level of abstraction	by functional simulation
		limited capabilities of the simulation tools	by plausibility checking of the simulation results
		common-cause faults (common data base for implementation and simulation)	additional validation later in design flow
	Use of Low Level Hardware Description Languages		by functional simulation (build-in or third party)
		limited capabilities of the simulation tools	by plausibility checking of the simulation results
		common-cause faults (common data base for implementation and simulation)	additional validation later in design flow
	Use of Hardware Description Languages, e. g. (V)HDL		by functional simulation
		poor design methodology (limited testability, timing critical (asynchronous) constructs)	by code review some problems are also revealed automatically, later in the design process.
		wide variety of different language constructs (with impact on synthesis results)	code review
	High Level Design Entry (same scope as “semi-formal” methods in specification phase) automated code generation		by functional simulation in the high level environment
		weak semantics of the input language	by review of the generated code
			by extended functional simulation of the generated code by automatic compare of the simulation results against the behaviour of the high level description
		faults during code generation quality and reproducibility of the generated code	by extended functional simulation of the generated code by automatic compare of the simulation results against the behaviour of the high level description
		validation only within high level entry tool (e. g. build-in simulator)	by functional simulation of the generated code, using an independent tool

-

Design Description	Use of “Soft Cores” or “Macro Blocks”		by functional simulation of the interaction with the surrounding blocks
		concentration on the interaction with the surrounding blocks	by functional simulation of the core or macro itself
		vendor dependent quality (correctness) of the core	by review by functional simulation
		encrypted or pre-compiled (“black box”)	by expanded functional simulation
	Schematic Entry		by review
		low level of abstraction (description at gate level)	by functional simulation
		use of macro blocks	by functional simulation
	all types of design entry	functional deviation from specification	by functional simulation (manual compare against specification) by (automated) cross check against specification
		partial verification, insufficient quality of the test cases	review of the test cases semi-formal methods to ensure coverage of the test cases

-

Implementation I	Conversion from Schematic to Netlist / Design Database		none; “correct by construction”
		(semantic) faults during conversion	by simulation (manual check against specification) by simulation (automated check against the simulation of the schematic) later in design flow
		no timing constraints	by additional tools later in design flow
	Conversion from High Level Entry to Netlist / Design Database		none; “correct by construction”
		(semantic) faults during conversion	by simulation (manual check against specification) by simulation (automated check against the simulation in the high level environment) later in design flow
		no timing constraints	by additional tools later in design flow
	Synthesis		none; “correct by construction”
		faults during synthesis process (resulting in functional discrepancies)	by automated cross check of the gate level simulation against the functional simulation (RTL)
		differences between the behaviour prior and post synthesis (related to poor design style or methodology)	by code review by automated cross check of the gate level simulation against the functional simulation (RTL)
		high complexity of the software and algorithms used for synthesis	by build-in checks by extended simulation of the results
		inappropriate timing	by gate level simulation with timing information by (static) timing analysis with independent tool
	Test Insertion		none; “correct by construction”
		fault, leading to modified functionality	by automated cross check of the simulation post vs. prior test insertion by formal equivalence check
		modified timing	by gate level simulation with timing information by (static) timing analysis
		wrong coverage figures	by fault simulation with independent tool

-

Implementation I	Use of “Generated Cores” or “Hard Cores”		none; “correct by construction”
		violation of design rules	later in design flow, by DRC
		mismatch between simulation model and behaviour of generated core	by using qualified generators or qualified core cells later, during production test or in circuit test
		conversion between technologies	by DRC by netlist and timing extraction, plus extended gate level simulation
	all implementation methodologies	faults in library (common cause fault for synthesis and simulation)	by using qualified or “proven in use” libraries
		faults in electrical or design rule set of the semiconductor vendor	by using qualified or “proven in use” information
		manual interference, manipulation of intermediate or final netlist or design database	by automated cross check of the simulation post vs. prior manipulation by formal equivalence check (if possible)
		partial verification, insufficient quality of the test cases used for manual or automated cross checks	by review of the test cases by semi-formal methods to ensure coverage of the test cases

-

Implementation II	Device Fitter		none; “correct by construction”
		in-circuit verification only	by extended (documented) in-circuit tests by additional simulation
		build-in simulator tools	by in-circuit tests by cross check with independent simulator
		timing violation	by review (PLD only, guaranteed for strict synchronous designs) by timing analysis (automated or manual)
		faults in library (common cause fault for fitter and simulation)	by using “proven in use” devices and environment by in-circuit tests
	Place & Route (FPGA)		none; “correct by construction”
		functional mismatch due to faults in P&R tool	by gate level simulation (post P&R netlist vs. prior P&R)
		timing violation	by gate level simulation (post P&R netlist and timing) by static timing analysis
		bitstream generation (FPGA only)	by in-circuit test
	Place & Route (Gate Array)		none; “correct by construction”
		functional mismatch due to faults in P&R tool	by gate level simulation (post P&R netlist vs. prior P&R) by LVS (if supported) later in design flow (production test)
		timing violation	by gate level simulation (post P&R netlist and timing) by static timing analysis
		design rule violations	by DRC
	Layout (ASIC)		none; “correct by construction”
		functional mismatch due to faults in layout process	by gate level simulation (post layout vs. pre layout) by LVS
		timing violation	by gate level simulation (post P&R netlist and timing) by static timing analysis
		design rule violations	by DRC

-

Production	programming of non-volatile devices		none; “correct by construction”
		invalid programming	by readback of the programmed information by parameter testing during the program cycle (e. g. resistance measurement)
		functional deviation (unrevealed device faults)	by running production test pattern by in-circuit test (all devices!)
	volatile devices		none; “correct by construction”
		corrupted bitstream (during download)	by checksum (if supported)
		functional deviation (after download)	by additional in-circuit measures
	mask generation (ASIC, Gate Array)		none; “correct by construction”
		faults during mask generation	by manual inspection by compare (two mask sets required) later in design flow (production test)
	production test (ASIC, Gate Array)		by production test (running test pattern)
		process variations	by inspection of critical paths by measurement of characteristically parameters
Post Production			by running set of standard validation tests, in addition to the pre-validation tests done during design and implementation phases.

Table 5: Fault Revealing in Design Flow

-

6 Validation Tests for Complex Components

To categorise the validation test sets for complex components, two parameters have to be considered:

- Safety Category, based on EN 954-1
- Complexity of the component

From these two parameters, the Safety Category is already clearly defined in EN 954-1. To categorise the complexity of a component, the following – indirect, based on “testability” – classification is used:

- A component is of ***low*** test complexity if it is adequate to run the standard validation tests on the final component (post production phase), and to reach the validation test coverage defined in Table 1.
- A component if of ***medium*** test complexity if running the standard validation tests on the final component achieves a maximum test coverage for at least one test that is one level less than required (e. g. “medium” coverage of functional testing instead of the required “high” coverage).
- A component is of ***high*** test complexity if running the standard validation tests on the final component achieves a maximum coverage for at least one test that is two or more level less than required (e. g. “low” coverage of functional testing instead of the required “high” coverage).

6.1 Testability and Complexity

To a certain extend, it is possible to use the “testability” rating from above as a mean to categorise the functional and structural *complexity* of a device or system. For example, a “simple” component, e.g. a member of the 74XX or 40XX TTL or CMOS series has a very limited functionality which makes it possible to do some functional tests and to achieve 100% test coverage. A more sophisticated component, like an embedded 8 bit micro controller may not be fully functional testable, due to practical limitations (time and effort required to create adequate functional tests); the test coverage might not be sufficient to fulfil the requirements from Table 1. In this situation, additional measures are required to fill the gap between achieved and

-

required test coverage. These additional measures may be of non technical nature, for example claiming “proven in use” for this device or may required additional verification / validation steps carried out during the design process. The later approach is detailed in the following chapter.

In most cases, the relation between “testability” and “complexity is bi-directional. This means that components with “low test complexity” has a “low functional complexity” and vice versa. Components with limited “testability” usually components with medium complexity and components with high complexity result usually in insufficient “testability”.

This bi-directional relation between “testability” and “complexity” does not necessarily exist in every case, so we use this classification scheme only to quantify validation tests, not to introduce a new complexity metrics. Introducing a new complexity seems promising right now, but this would require additional work, and is beyond the scope of WP3.3 or the STSARCES project.

6.2 Validation Tests carried out during Design

6.2.1 Components with Low Test Complexity

For components with low test complexity (good “testability”), it is adequate to run the standard validation test set after component production. This is a direct implication of how the term “low test complexity” is defined at the beginning of this chapter. The result is shown in Table 6 (which, in this case, is equivalent to Table 1). No validation tests during the design process are required.

-

Technique / measure	Cat 1,2		Cat 3		Cat 4	
	During Design Flow	Post Production	During Design Flow	Post Production	During Design Flow	Post Production
Functional testing	– –	HR high	– –	HR high	– –	HR high
Functional testing under environmental conditions	– –	HR high	– –	HR high	– –	HR high
Interference immunity testing	– –	HR high	– –	HR high	– –	HR high
Fault injection testing	– –	HR high	– –	HR high	– –	HR high
Expanded functional testing	– –	– low	– –	HR low	– –	HR high
Surge immunity testing	– –	– low	– –	– low	– –	– medium
Black box testing	– –	R low	– –	R low	– –	R medium
Statistical testing	– –	– low	– –	– low	– –	R medium
“Worst case” testing	– –	– low	– –	– low	– –	R medium

Table 6: Validation Tests for Components with Low Test Complexity

6.2.2 Components with Medium Test Complexity

For components with medium test complexity, some validation tests need to be run during the design and implementation phases. The required test set and the required coverage is given in Table 7; Chapter 5.2.2 “Validation Tests & Phase Model” shows at which point in the design process it is advised to run the individual tests (for details, see Table 3).

Additional verification loops are required. See chapter 6.3.

-

Technique / measure	Cat 1,2		Cat 3		Cat 4	
	During Design Flow	**Post Production**	**During Design Flow**	**Post Production**	**During Design Flow**	**Post Production**
Functional testing	HR high		HR high		HR high	
	high	medium	high	medium	high	medium
Functional testing under environmental conditions	HR high		HR high		HR high	
	high	medium	high	medium	high	medium
Interference immunity testing	HR high		HR high		HR high	
	–	high	–	high	–	high
Fault injection testing	HR high		HR high		HR high	
	high	medium	high	medium	high	medium
Expanded functional testing	– low		HR low		HR high	
	low	low	low	low	high	medium
Surge immunity testing	– low		– low		– medium	
	–	low	–	low	medium	low
Black box testing	R low		R low		R medium	
	–	low	–	low	medium	low
Statistical testing	– low		– low		R medium	
	low	low	low	low	medium	low
“Worst case” testing	– low		– low		R medium	
	low	low	low	low	medium	low

Table 7: Validation Tests for Components with Medium Test Complexity

6.2.3 Components with High Test Complexity

For components with high test complexity, a reasonable number validation tests need to be run during the design and implementation phases. For safety reasons, it is not useful to give general recommendations about the required test set and the required coverage for components with high test complexity without detailed knowledge about the component and its intended use.

-

6.3 Implementation / Verification Loops

Moving the validation testing to an earlier point in the design flow, the subsequent steps need to be more thorough verified, to ensure that the results of the validation are still valid for the final component. All listed verification steps need to be carried out that are required for an uninterrupted chain of cross-checks, starting at the validation test in the design process and ending at the final component. The coverage for each step needs to be at least as high as the coverage for the validation test itself (Table 7). If more than one verification method is listed, at least one (or any meaningful combination) has to be used.

Table 8 sums up the verification tasks from Table 5.

-

Phase	Implementation Step	Verification Step
Specifi-cation	– – –	– – –
Design Description	Code generation from High Level Design Description	functional simulation of the resulting code equivalence check, using automatic compare of the simulation results in the High level environment vs. the simulation results of the generated code
	Use of “Soft Cores” or “Macro Blocks”	functional simulation of the core or macro + code review extended functional simulation of the core or macro (if no source code is available)

-

Implementation I	Conversion from Schematic to netlist / design database	simulation of the resulting netlist (manual check against specification) simulation of the resulting netlist (equivalence check against behaviour of schematic)
	Conversion from High Level Entry to netlist / design database	simulation of the resulting netlist (manual check against specification) simulation of the resulting netlist (equivalence check against behaviour of high level description)
	Synthesis	simulation of the resulting gate level netlist (manual check against specification) [only if low coverage is required] simulation of the resulting gate level netlist (equivalence check against behaviour of (V)HDL source code)
		simulation of the gate level netlist with timing information, to verify timing constraints static timing analysis
	Test Insertion	simulation of the resulting gate level netlist (equivalence check against the netlist prior to test insertion) formal equivalence check
		simulation of the gate level netlist with timing information, to verify timing constraints static timing analysis
	Use of “Generated Cores” or “Hard Cores”	DRC (design rule check) netlist extraction, extended simulation of the resulting netlist
		netlist and timing extraction, extended simulation of the netlist with timing information, to verify timing constraints netlist and timing extraction, static timing analysis

-

Implementation II	Device Fitter	extended in-circuit test export into a netlist, simulation
	Place & Route (FPGA)	export of P&R database into a netlist, extended simulation of the netlist export of P&R database into a netlist, simulation (equivalence check against the pre P&R netlist) export of P&R database into a netlist, formal equivalence check
		export of P&R database into a netlist and into a timing file, gate level simulation with timing, to verify timing constraints) export of P&R database into a netlist and into a timing file, static timing analysis
	Place & Route (FPGA)	export of P&R database into a netlist, extended simulation of the netlist export of P&R database into a netlist, simulation (equivalence check against the pre P&R netlist) export of P&R database into a netlist, formal equivalence check
		export of P&R database into a netlist and into a timing file, gate level simulation with timing, to verify timing constraints) export of P&R database into a netlist and into a timing file, static timing analysis
		DRC (design rule check)
		LVS (layout vs. schematic check)
	Layout (ASIC)	netlist extraction from layout, extended simulation of the netlist netlist extraction from layout, simulation (equivalence check against the pre P&R netlist) netlist extraction from layout, formal equivalence check
		netlist and timing extraction from layout, gate level simulation with timing, to verify timing constraints) netlist and timing extraction from layout, static timing analysis
		DRC (design rule check)
		LVS (layout vs. schematic check)

-

Production	non-volatile devices (PLD, CPLD, FPGA)	readback of programmed device, parameter testing running “production test” pattern on final device
	volatile devices	readback of configuration PROM
	mask generation (ASIC)	mask inspection mask compare
	defects	running “production test” pattern on final devices
	process variations	timing measurement (on ASIC tester) of critical / characteristic paths measurement of typical process parameters

Table 8: Verification Tasks

-

7 Conclusion

As part of the work on Work Package 3.3 “Safety Validation of Complex Components – Validation Tests”, several state of the art validation test methods that are in use for complex or semi-complex components where evaluated and assessed. Typical work flows for the design of PLDs, FPGAs and (cell based) ASICs were used as reference to identify possible safety hazards in the design and development process of such complex (hardware) components.

As a result of the work on Work Package 3.3, guidelines for suitable *validation tests* – that consist of a number of *validation tasks* that need to be carried out during the design and development process – were proposed. This enables the designer of this type of components to provide the objective evidence that the functional and the safety objectives for the complex component under consideration are met.

-

Appendix A: Safety Validation Methods

This chapter gives some additional information on the different “Safety Validation Methods” mentioned in the main part of this report. The main base of these descriptions is the IEC 61508.

A.1 Functional testing

Functional testing is used to reveal failures during the specification and design phases and to avoid failures during implementation and the integration of software and hardware.

A.1.1 How to proceed ?

During the functional tests, reviews are carried out to see whether the specified characteristics of the system have been achieved. The system is given input data which adequately characterises the normally expected operation. The outputs are observed and their response is compared with that given by the specification. Deviations from the specification and indications of an incomplete specification are documented.

Functional testing of electronic components – designed for a multi-channel architecture – is carried out by testing the manufactured components against pre-validated partner components. In addition to this, it is recommended to test the manufactured components in combination with other partner components of the same batch, in order to reveal common mode faults which would otherwise have remained masked.

A.1.2 Comments based on practical use

The method of functional testing is one of the most popular methods during the past years to deal with safety relevant programmable electronic systems. But with increasing complexity of the components used for electronic systems the effectiveness of the coverage of detection of faults and defects of these complex circuits is decreasing. It is not possible to test all logic combinations of a complex circuit. Also a subset of tests delivers an insufficient result.

-

A.1.3 Applicability for complex components

Beside the problem of pure complexity that makes is practical impossible to do a adequate functional testing, the use of high complex components raises additional problems:

controllability: During functional testing, each complex component has to be treated as a special type of “black box”. Although all details about this “black box” may be specified and known to the tester, it is not possible to go “inside” the component to do a functional testing of the individual building blocks. Thus it might not be possible to check most of the functional details that are not directly controllable from the components boundary. More disadvantageous, not even all safety functions may be tested, especially those part that deal with potential faults (e. g. using self-testing logic or redundancy) might not be tested because they may not be activated during normal operation.

observability: Internal states of a integrated component may not be fully visible for the outside world. Thus the behaviour of the single component or a complex system may be non-deterministic from the testers point of view. This type of “random” behaviour may be triggered by a special sequence of events that might not be reproducible nor be classified with respect to the safety function.

repercussion: The test setup required for functional testing itself may have a serious impact on the system under test. For example, it might not be possible to run the system at full speed (because an emulator is use instead of the on-board CPU) or it is necessary to attach probes – that represent an additional capacitive and inductive load – to trace on-board signals.

A.1.4 Conclusion

From the practical experience, functional testing is an very effective method for validation testing, but only if the system under test has a limited complexity. Functional testing is applicable for complex systems and components, but high complex monolithic systems need to be partitioned into smaller, more manageable units to benefit from a functional test. Moreover, “virtual” functional testing, e. g. using simulation during the design process, may provide a very precise information about the behaviour of the system under special modes of operation – even under those

-

conditions that might not be checked during a “real” functional test, due to the mentioned lack of controllability.

A.2 Functional testing under environmental conditions

This method provides that the safety-related system is designed to operate under the specified environmental conditions and that it is protected against typical environmental influences.

A.2.1 How to proceed ?

The system is put under various environmental conditions (for example according to the standards in the IEC 60068 series or the IEC 61000 series), during which the normal operation and the safety functions are assessed.

A.2.2 Comments based on practical use

The method of functional testing under environmental conditions is a very good method to check a subset of functions during or after exposure to environmental stress (climatic, mechanic as well as electromagnetic stress etc.). But it is not possible to test all logic combinations of a complex circuit. This method can be understood as an addition to functional testing as already commented.

A.2.3 Applicability for complex components

It is a well known fact that environmental stress (e. g. high temperature) has a statistical impact on the expected lifetime of different types of components. Based on long term experience and process characterisation, many of the operating conditions (e. g. supply voltage, ambient temperature) required for reliable and long-term stable operation are known in advance.

A.2.4 Conclusion

In addition to the functional testing under environmental conditions, the functionality and behaviour of a complex component under environmental conditions may be estimated in advance, based on known characteristics of the devices physics and the manufacturing process.

-

A.3 Interference surge immunity testing

To check the capability of the safety-related system to handle peak loads, the method of interference surge immunity testing is to be done.

A.3.1 How to proceed ?

The system is loaded with a typical application program and all the peripheral lines (all digital, analogue and serial interfaces as well as the bus connections and power supply) are subjected to standard noise signals. In order to obtain a quantitative statement, it is sensible to approach the surge limit carefully. The chosen class of noise is not complied with if the function fails.

A.3.2 Comments based on practical use

This method is one of the basic methods to ascertain, that the programmable electronic system is able to work under special environmental conditions (especially electromagnetic conditions) without loss of the safety function.

A.3.3 Applicability for complex components

The main focus of interference surge immunity testing is on external interfaces and on interconnections. Thus, surge immunity is primary a problem to be addressed on board level – where additional protection circuitry might be required – primarily independent of the complexity of the components used to implement the core functionality.

A.3.4 Conclusion

Surge immunity testing is applicable independent of the type of components used on a board. But, high complex components might demand a higher level of external protection circuitry, due to lower immunity to noise and voltage surge.

A.4 Fault injection testing

Fault injection testing is used to introduce or simulate faults in the system hardware and document the responses.

-

A.4.1 How to proceed ?

This is a qualitative method of assessing dependability. Preferably, detailed functional block, circuit and wiring diagrams are used in order to describe the location and type of fault and how it is introduced. For example: power can be cut from various modules; power, bus or address lines can be open/short circuited; components or their ports can be opened or shorted; relays can fail to close or open, or do it at the wrong time, etc. Resulting system failures are classified, as in tables I and II of IEC 60812, for example. In principle, single steady state faults are introduced. However, in case that a fault is not revealed by the built-in diagnostic tests or otherwise does not become evident, it can be left in the system and the effect of a second fault must be considered. The number of faults can easily increase to hundreds. The work is done by a multidisciplinary team and the vendor of the system should be present and consulted. The mean time between failure for faults that have grave consequences should be calculated or estimated. If the calculated time is low, modifications should be made.

A.4.2 Comments based on practical use

Fault injection testing is mandatory, because a clear reaction of the system or the component on a fault or a faulty state only can be available by fault injection. The theoretical base of fault injection testing normally is a failure mode and effects analysis (FMEA) either on system level or on levels of analysis lower than the system level. The lowest level is the component level. The FMEA is one of the best theoretical instruments to analyse system or component states and the reaction of the system or subsystem or component to faults. It is an essential need, that tests that are based on the method of fault injection testing are defined as a result of a theoretical / analytical method like the FMEA, FTA or ETA, Cause Consequence Diagrams, Worst Case Analysis, et cetera. Only with this procedure, the effectiveness of testing is guaranteed. With this analytical methods the possible faults of the systems or components were analysed and the effects of faults on different system levels are to be considered systematically.

-

A.4.3 Applicability for complex components

As described above, the lowest level of an FMEA is the component level. But, in the context of complex components, each such component (e. g. a microprocessor with on-board RAM and program ROM and a full custom ASIC) may represent a full self-contained, independent sub-system. Treating such a component as a single, indivisible entity in an FMEA might render the complete FMEA useless. Moreover, during fault injection testing, it might be impossible to reach all relevant internal states and nodes from the inputs of the complex component under test.

A.4.4 Conclusion

As for functional testing, it is also required for FMEA and fault injection testing to move “beyond to surface” of a complex component. Due to limited controllability – it might not be possible to inject faults into a component, even with high sophisticated test equipment – fault injection testing has to move to an earlier stage in the design process. The most promising approach is to use fault injection testing together with functional simulation.

A.5 Worst case testing

To test the cases specified during worst case analysis.

A.5.1 How to proceed ?

The operational capacity of the system and the component dimensioning is tested under worst case conditions. The environmental conditions are changed to their highest permissible marginal values. The most essential responses of the system are inspected and compared with the specification.

A.5.2 Comments based on practical use

Worst case testing is not always possible, because it is very difficult to define the limits, where the equipment under test is not destroyed or damaged with long time defects. Normally worst case testing is done with a couple of prototypes after running tests under normal specified conditions for the system or the component. After this normal condition testing the prototypes will be tested slowly to their limits to define

-

the real limits of use. After worst case testing the equipment under test is analysed closely. The equipment under test normally is damaged when the worst case test was done successfully.

A.5.3 Applicability for complex components

If the defects due to worst case testing are assumed to be equally distributed, worst case testing of a complex component will result in random failure modes. To get an expressive result – a classification or numerical distribution – for the portion of safety related faults, a quite large number of systems will be required for worst case testing. This is not acceptable, not only from the commercial point of view.

As already stated in the chapter “Functional testing under environmental conditions”, a priory knowledge from experience and process characterisation, may help to find out the absolute maximum conditions for worst case testing. Static analysis might help to characterise the actual behaviour under worst case stress and clearly show the weakest points of the component, without the necessity to run a destructive test.

A.5.4 Conclusion

A priory knowledge about the behaviour of a component under stress is useful, both for the improvement of the component itself as well as for a prediction of the outcome of a worst case test. Used at an early point in the design process, it might help to reveal potential problems that would otherwise first show up during worst case testing. Moreover, a prediction about the expected behaviour might help to focus on the “right” part of the system during worst case test.

A.6 Expanded functional testing

Used to reveal failures during the specification, design and development phases. Also used to check the behaviour of the safety-related system in the event of rare or unspecified inputs.

A.6.1 How to proceed ?

Expanded functional testing reviews the functional behaviour of the safety-related system in response to input conditions which are expected to occur only rarely (for

-

example major failure), or which are outside the specification of the safety-related system (for example incorrect operation). For rare conditions, the observed behaviour of the safety-related system is compared with the specification. Where the response of the safety-related system is not specified, one should check that the plant safety is preserved by the observed response.

A.6.2 Comments based on practical use

This method is done for testing the limits of normal use and to define the system reactions in case of unknown stress and unknown fault combinations. For safety related complex components expanded functional testing is mandatory on prototypes.

A.6.3 Applicability for complex components

As for regular functional testing, expanded functional testing will not have an adequate coverage of a complex component's total functionality, nor of the safety related subset of this functionality. To catch up with the complexity issue, it is necessary to divide the whole functionality into smaller, more manageable units. Because this is not possible looking at component level, this partitioning needs to be done at an earlier stage of the design process, e. g. using extended functional simulation at module level.

A.6.4 Conclusion

Expanded functional testing is only possible at the boundary of the final component; the coverage of the expanded functional test for internal building blocks of the component will be unsatisfactory low in most cases. Again, it is necessary to go "beyond the surface" of the component and to do the expanded functional testing earlier in the design process, using an adequate "functional model".

-

Appendix B: Technology Overview

The following paragraphs give an overview of typical products and design methodologies for integrated circuits. The number of parties involved in the design and validation process varies as well as the responsibility for the work packages within the design flow.

	PLD CPLD	FPGA	gate array	cell based ASIC	core based ASIC	full custom ASIC	standard IC
functional specification	C	C	C	C	C	C	V
implemen-tation	C	C	D	D	D, M (1)	D (1)	V
place & route, layout	V	C	V	D	D, M (1)	D (1)	V
wafer production	V	V	V	V	V	V	V
packaging	V	V	V	V	V	V	V
test (2)	V, C (3)	V, C (3)	V, D (4)	V, D (4)	V, D (4)	V	V

Table 9: Overview Integrated Circuits

The following notation is used in Table 9:

responsibilities	V	IC / ASIC vendor (manufacturer)
	C	end customer, system and application development
	D	ASIC design centre
	M	macro core (pre-designed functional blocks) vendor
Notes	(1)	ASIC design centre of the silicon vendor or independent design centre (third party)
	(2)	For standard IC and ASIC design, “test” denotes the production test that ensure integrity of the device prior to shipping.
	(3)	in this case: system integration test, in addition to production test for the un-programmed devices
	(4)	production test, done during manufacturing process by ASIC Vendor; based on test patter generated and approved by D

B.1 Standard IC

Manufactured in large quantities and applied for different applications. Functionality, validation, production and production test are solely in the hand of the semiconductor

-

vendor. Manual manipulations and optimisations at layout level are frequently used to reduce required area. Not designed for safety-related systems, fault avoidance during the design process is only adequate for standard products. Frequent changes in production process, process technology and layout are likely for cost and yield optimisation. Number of components manufactured using a certain process or mask revision are not publicly known.

B.2 Full Custom ASIC

Application Specific Integrated Circuit. Design and production similar to standard IC, with functionality defined by end customer.

B.3 Core Based ASIC

Based on pre-layouted or generated macro cores, connected by additional logic. Examples for pre-layouted macros are standard microprocessor cores, peripheral components, communication interfaces, analogue blocks, special function I/O cells. Examples for generated macros include embedded RAM, ROM, EEPROM or FLASH. Generated blocks are assumed to be „correct by construction“, based on design rules. Pre-layouted or generated macros are process specific but may be ported to different technologies. In most cases, the macro cores are not identical to the original discrete off-the-shelf components (different process, provided by a third party).

B.4 Cell Based ASIC

Based on logic primitives (like AND, OR, Flip-Flop, Latch) taken from a cell library. The gate-level netlist containing the logic primitives and the interconnections is usually created from a high level hardware description language (VHDL, Verilog) using synthesis tools. The functional and timing characteristics of the logic primitives is characterised in the cell library; these parameters are used to drive the synthesis tool and are also used for simulation. In addition, layout tools are used to place the cells and to route the interconnects.

-

B.5 Gate Array

Pre-manufactured silicon „masters“ with a fixed number of cells are the common starting point for different components. The functionality is defined by the interconnection matrix (metal layer) between the pre-manufactured cells. The design process it very similar to that of a cell based ASIC, while the layout step is replaced by a routing step to connect the already existing cells.

B.6 FPGA

Field Programmable Gate Array. Standard IC, using one-time programmable or re-programmable elements to define the connection between functional blocks and to configure the functionality of the individual blocks. It is not possible to test one-time programmable FPGAs completely during production due to the nature of the programmable element.

B.7 PLD

Programmable Logic Device. Standard IC, with low to medium complexity, using one-time programmable or electrical erasable elements (“fuse”) to define combinatorial logic – typical based on AND / OR product terms – and configurable storage elements. Predictable timing and guaranteed maximum operating frequency in synchronous design due to regular structure.

B.8 CPLD

Complex PLD. Multiple PLD-like blocks on a single chip, connected by a programmable interconnection matrix (crossbar). The programmable logic element is re-programmable (EPROM or EEPROM) in most cases.

B.9 MCM

Multi Chip Module. Multiply chips (dies) and passive components mounted on a common substrate and assembled into a single package. In most cases, package and outline is similar a standard IC. The chips (dies) use for MCM production are usually pre-validated, but not finally characterised. Thus, testing under environmental conditions needs to be done at MCM level.

-

MCM is primarily a different packaging technology. Design methodology for the individual parts of the MCM is mostly identical to the design methodology for system build on conventional printed circuit boards. Therefore, MCM are not further discussed in this report.

B.10 COB

Chip On Board. Instead of using chips (dies) in conventional packages, the die is bonded directly on the printed circuit board and hermetically sealed afterwards.

As mentioned for MCM, COB is primarily a different packaging technology, thus no further discussed in this report.

-

Appendix C: Complexity Metrics

To clarify the term „complex component“ in the context of safety validation, it is useful to introduce a classification scheme for complex components. A classification makes it possible to tag every component class with an individual set of required or recommended safety validation test. In most cases, the set of validation tests assignable to each class will be only a subset of all safety validation tests considered in WP 3.3, leaving out those tests that are either not applicable or not meaningful for the component class under consideration.

The classification of complex components may be done according to different metrics. Possible metrics include, but are not limited to:

- **structural complexity**, e. g. measured in the number of bundled components or the number of integrated gate equivalents
- **functional complexity**, e. g. measured in the number of functional requirements assigned to the component or the extent of the component’s specification
- **technology**, including semiconductor process, packaging, mounting and assembly technologies
- **field experience**

Structural and functional complexity should be clearly distinguished. For example, state-of-the-art RAM chips are among the components with the highest structural complexity, integrating millions of single-bit memory cells in a single chip. But, on the other hand, the functional complexity of a RAM is very low – its functionality may be specified in a few statements. The consequences for safety validation test are, that, due to the low functional complexity and the regular structure, black box testing at component level, e.g. with algorithms described in IEC 61508, is adequate and ensures high coverage.

C.1 Structural Complexity

Advances in semiconductor process technology are the driving factors for increased structural complexity of components. The typical structural complexity doubles with every new process generation.

-

Due to the number of integrated circuitry and interconnections, all possible failure modes of most complex component are not known nor is it possible to analyse the effects of the known failure modes with respect to the module or board where this component is used. Automatic tools for fault coverage analysis or fault injection testing are already used during the design process of complex components. Typically, these tools are well suited for fault coverage calculation using a given or automatically constructed set of test patterns. The usability of such tools for failure mode examinations has to be evaluated. Moreover, even for the fault detection coverage, additional work is required to make coverage figures estimated for *functional* test (e.g. self-test code executed by a microprocessor) comparable to coverage figures calculated based on the actual *structural* information (like layout or netlist). New fault models are required due to the advances in semiconductor technology as the minimum feature size has moved far beyond one micron, resulting in fault scenarios not covered by conventional stuck-at fault models.

C.2 Functional Complexity

As shown in Figure 2, more and more functionality may be integrated into a single component. In every phase shown in the figure, functionality implemented at module or board level is packed into a single components in the next generation. The total complexity rises by several orders of magnitude.

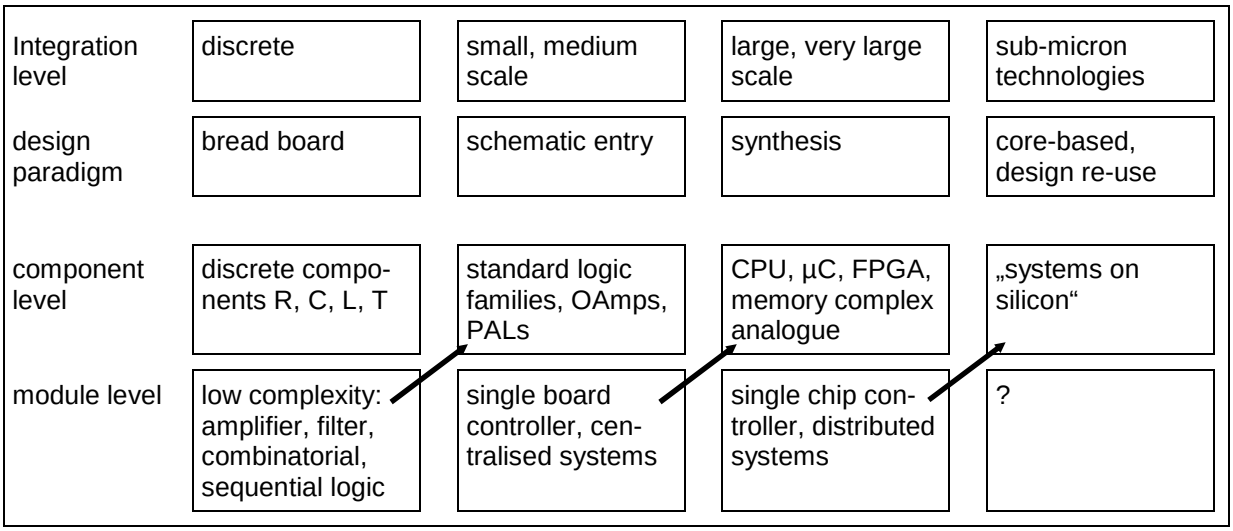


Figure 2: Integration Stages

-

For safety validation in the context of complex components, it is no longer adequate to consider components as atomic building blocks of circuit modules or boards. Instead of this „black box“ approach, it is necessary to move beyond the component level to perform meaningful and adequate validation test. It is obvious that this kind of testing is not possible after the integration. New methods and guidelines for functional testing during the design and integration process are required.

C.3 Technology

Different technologies used for “complex components” are already discussed in Appendix B: “Technology Overview”.

C.4 Field Experience

The definitions of IEC 61508 (part 2) for class A and B components implies that „... field experience should be based on at least 100.000 hours operating time over a period of two years with 10 systems in different applications.“ Especially for complex standard components, it is not known to the end user whether the devices that are actual used on the circuit board are manufactured for the required period of time with the current mask revision and on the current process line. Even if the standard component is available for many years, modifications during that period of time are most likely, contradicting the requirements laid down in IEC 61508.

For complex application specific integrated circuits (ASICs), the terms „experience“ or „proven in use“ should be clarified and related to the different inputs for the design process:

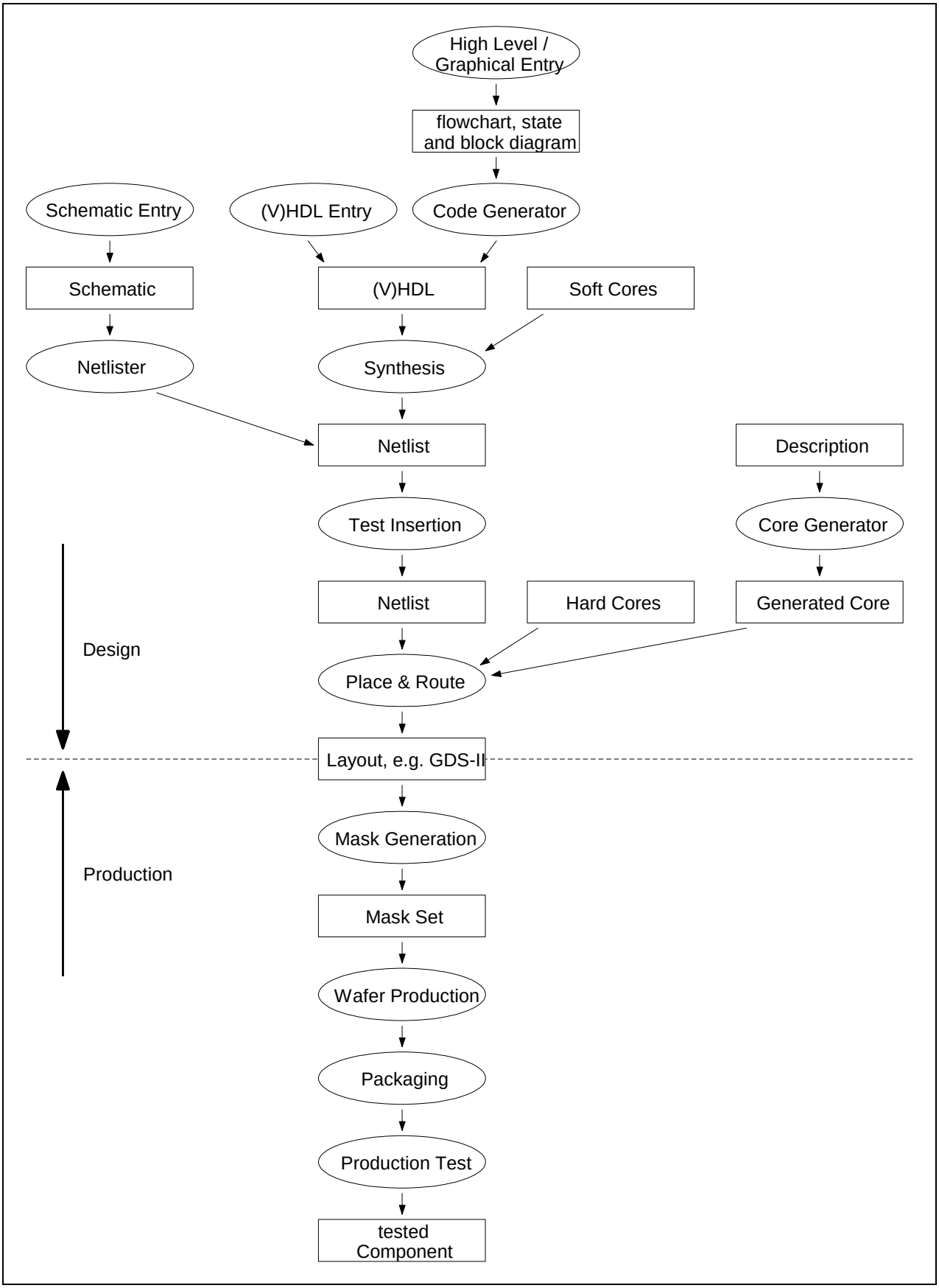
- process technology
- design rules for cell placement, interconnect and layout
- pre-layouted or generated macro cores
- cell libraries, including layout information and simulation models
- soft macros
- design tools: layout, synthesis, simulation

-

Appendix D: ASIC Design Flow

A simplified design flow for application specific integrated circuits (ASICs) and Gate Arrays is given in the following figure. The work packages shown in the design flow and the validation tests are listed in the following paragraphs.

-


High Level / Graphical Entry
flowchart, state and block diagram
Schematic Entry
(V)HDL Entry
Code Generator
Schematic
(V)HDL
Soft Cores
Netlister
Synthesis
Netlist
Description
Test Insertion
Core Generator
Netlist
Hard Cores
Generated Core
Design
Place & Route
Layout, e.g. GDS-II
Mask Generation
Production
Mask Set
Wafer Production
Packaging
Production Test
tested Component

-

Figure 3: Simplified Gate Array / ASIC Design Flow

D.1 Design Entry

D.1.1 Hardware Description Languages

Design description using a hardware description language like VHDL or Verilog[4]. This is most common hardware description methodology used today in ASIC and Gate Array design. Both languages are defined by IEEE standards and are assumed to satisfy the requirements for “high level programming languages” for safety related E/E/PE systems stated in IEC 61508.

The hardware description language may be used both for design description and for functional models or “test benches”. When used for design description, only a subset of the language may be used; this synthesiseable code is often referred to as RTL (“register transfer level”) code. Non synthesiseable code, adequate for functional models and test benches is called “behavioural” code.

D.1.1.1 Verification of the Results

Verification of the functionality is done using standard (V)HDL simulators. Simulation is done at (V)HDL source code level, ensuring the correct sequence of events but not the actual timing behaviour. Test scenarios and test case are derived from the specification requirements and have to be implemented manually, using the hardware description language.

D.1.1.2 Potential safety hazards

– Simulated behaviour at (V)HDL source code level (RTL) may differ from behaviour at gate level. For example, in an RTL description, a VHDL process may be defined to be sensitive only to a subset of its input signals. After synthesis, at gate level, the generated circuitry is always sensitive to every input signal.

[4] The term (V)HDL is used in this paper to denote either the VHDL or Verilog hardware description language.

-

– Wide variety of different language constructs. As for safety related software, only a subset of the language should be used due to potential limitation of the synthesis process and to improve readability.

– coverage of test scenarios and test cases.

D.1.2 High Level Design Entry

Comparatively easy to use graphical tools are used for high level design entry (flowcharts, state diagrams, spreadsheets, block diagrams); this provides a very descriptive method for design entry, with a high degree of self-documentation. Additionally, it is possible to use this methodology already during specification. The tools are able to create synthesiseable (V)HDL code from the graphical description. In some cases the transformation is bi-directional, able to create a graphic representation for (V)HDL source code, too.

D.1.2.1 Verification of the Results

Verification of the functionality is usually done by simulation, either with a simulator working inside the graphical tool or with a standard (V)HDL simulator after code generation, with back-annotation and visualisation of the simulation results in the front end tool.

Test scenarios and test cases are derived from the specification requirements and have to be implemented manually, in most cases in a tool specific environment and language.

D.1.2.2 Potential safety hazards

– Weak semantics of the input „language“. The (V)HDL code generated is only one possible representation of the functionality, leaving uncertainty about the implementation generated.

– The generated code may be hard to understand, e. g. during code reviews.

– Simulation exclusively in the graphical environment does not reveal faults introduced during the (V)HDL generation step.

-

– The quality of the test scenarios and test cases used during the verification may be not high enough.

D.1.3 Use of “Soft Cores” or “Macro Blocks”

“Soft Cores” are pre-designed – often parameterisable – blocks with a closed functionality, e. g. for multi-bit arithmetic (adder, multiplier, divider, etc.), commonly used interfaces, peripherals or even processor cores. In most cases, soft cores are used to build larger systems, to re-use already existing blocks and to speed up the design process.

D.1.3.1 Verification of the Results

Used “as is”, verified together with the blocks of the surrounding system.

D.1.3.2 Potential safety hazards

- Inadequate verification that concentrates on the interaction with the surrounding system only and does not verify the behaviour of the soft core or macro itself.
- Vendor-dependent quality of the soft core or macro libraries. Correctness is not guaranteed.
- Encrypted or pre-compiled, source code not available.

D.1.4 Schematic Entry

Schematic entry of the circuit, using primitives (single logic gates, Flip Flops) from a cell library or using macro functions (e.g. counters, standard logic components). The schematic may be translated directly into a corresponding gate-level netlist. For macros, a suitable gate-level representation is automatically substituted during the conversion process.

D.1.4.1 Verification of the Results

Verification of the functionality is done using standard simulators at gate level. Back-annotation of the results into the schematic is possible.

-

D.1.4.2 Potential safety hazards

– Old-fashioned design methodology, not used for larger designs in a state-of-the art design process due to the low level of abstraction, demanding the designer to generate the gate-level implementation of the required functionality manually.

– simulation results are depending on the simulation models stored in the macro block library.

D.2 Implementation

D.2.1 Synthesis

Automatic, constraints guided transformation of a (V)HDL description into a gate level netlist. The synthesis process is rather complex and is based on three different inputs:

– the (V)HDL description to define the functionality

– synthesis constraints (e.g. for path delays, area) to guide the selection of an appropriate implementation (out of all possible implementations that have the required functionality)

– a cell library as a collection of available target cells. Every cell in the library is characterised by its functionality and timing behaviour.

D.2.1.1 Verification of the Results

– internal housekeeping and checks during the synthesis process itself (automatically performed by synthesis tool)

– simulation of gate level netlist against the RTL reference model (functional equivalence)

– simulation of gate level netlist to verify timing constraints

– static timing analysis to verify timing constraints

D.2.1.2 Potential safety hazards

– functional discrepancy between (V)HDL source and gate level netlist due to

-

- language limitations (see (V)HDL Coding)
- faults during the synthesis process (caused by the synthesis tool)
- faults during manual interference in the synthesis process or manipulation of the netlist

  In general, these potential faults should be discovered during simulation of the gate level netlist against the behaviour of the RTL reference.

  It is important to note that simulation only reveals those faults actually covered by the test cases. Although it is desirable to re-run the complete set of validation tests done at RTL level after synthesis, in some cases this is not possible due to runtime restrictions or due to modifications in the module hierarchy (e.g. if several small modules are melted into a single module for the improvement of the synthesis results).

- faults in the cell library may cause discrepancies between the cell's actual functionality or timing behaviour and the behaviour of the model stored in the library. This may cause a "common cause failure" that is not revealed by simulation, because synthesis, simulation and static timing analysis are depending on information from the cell library. But, functional faults will be revealed during production test if the functional mismatch is testable and covered by the test pattern.
- very complex software and algorithms are used during the synthesis process. Due to the complexity and the ongoing development of the tools, it seems not possible nor desirable to certify a particular tool and ban the usage of not certified tools.

D.2.2 Conversion from Schematic to Gate Level Netlist ("Netlister")

For schematic entry, the tool-internal design database that represents the schematic must be translated into a gate level netlist. This process is similar to the synthesis process described before, but far less complex.

D.2.2.1 Verification of the Results

- simulation of gate level netlist to verify functionality, eventually with back-annotation into the original schematic

– simulation of gate level netlist to verify timing constraints

– static timing analysis to verify timing constraints (requires addition tools)

D.2.2.2 Potential safety hazards

– Functional discrepancy between the schematic and gate level netlist due to

  – faults during the conversion process

  – faults in the macro library, leading to a false implementation of the macro's functionality

  – faults during manual interference in the synthesis process or manipulation of the netlist

– No timing information in schematic, thus no timing constraints are respected in the translation process

D.2.3 Test Insertion

Automatic insertion of test structures into the netlist, like scan (for automatic test pattern generation, ATPG), boundary scan or build in self test (BIST). In addition to the scan insertion, a set of test vectors is generated during the test insertion process. Fault coverage, in most cases based on a „single-stuck-at" fault model is automatically calculated.

Test insertion, fault coverage analysis and fault simulations are primarily done to ensure testability of the chip after manufacturing, in other words to detect structural faults during the manufacturing process and guarantee the integrity of the manufactured devices after the production test. The analysis is *not* done to reveal the effects of faults with respect to the system.

D.2.3.1 Verification of the Results

– simulation of the netlist after test insertion against the behaviour of a reference model (netlist prior to test insertion or RTL source code) with respect to functionality and timing.

– Static timing analysis

-

– functional simulation of the ATPG test pattern set

– functional simulation of the boundary scan

– functional simulation of the BIST

– fault simulation (to check calculated coverage figures or to analyse coverage of functional patterns and BIST)

D.2.3.2 Potential safety hazards

Faults during test insertion (functionality, timing). These faults are revealed by simulating the behaviour of the netlist against a reference.

D.2.4 Generated Cores, Hard Cores

Regular structured macro cores, like RAM and ROM blocks, are usually generated separately and linked to the design database for use in the layout process. The generator provides two separate outputs: The pre-layouted macro core itself, directly useable for layout and a simulation model of the core for the gate level simulation.

“Hard Cores” are an other type of pre-layouted macro. They span the same functionality as “soft cores” (e. g. communication interfaces or peripherals, microprocessors), but are provided as already optimised, but technology-dependent, pre-layouted blocks.

D.2.4.1 Verification of the Results

– Use of a simulation model for functional simulations of the RTL description or the gate level netlist to ensure proper interactions with the macro core.

– Design rule check (DRC) for the generated layout of the core.

D.2.4.2 Potential safety hazards

– In most cases, the model used for simulation and the core are derived from the same source. But, besides this common origin, there is no further relation between the functionality of the simulation model and the functionality of the core. Thus, discrepancies between the two instances are possible.

-

– The design rules defined by the semiconductor vendor ensure adequate electrical characteristics and compliance to the process requirements. Even if the DRC does not detect violations, this does not guarantee correct functionality in any case. Thus, for example, faults in a core generator may not be revealed.

– Hard cores are not portable between different technologies. In some cases, it is possible to automatically convert the layout from one technology to an other. Faults during this process may not be revealed.

D.2.5 Place and Route / Layout

In a first step, the cells found in the final gate level netlist and the macro cores are placed on the chip. *Note*: This step is required for core and cell based designs only, for gate arrays, a regular placement of universal cells has already be done during the pre-production of the gate array master. In a second step, the interconnections are routed. In a third step, timing information are derived form the actual layout and back-annotated for post layout simulation.

In many cases, the place and route / layout step includes additional tasks like

– buffer sizing, adapting the output drive strength of individual gates to the actual wire load after layout

– clock tree synthesis, generating a skew-optimised clock distribution system.

D.2.5.1 Verification of the Results

– Simulation of the netlist after layout against the behaviour of the reference model (netlist or RTL source code) with respect to functionality and timing.

– Static timing analysis

– Design rule check (DRC) to guarantee the design rules dictated by the semiconductor vendor.

– Layout versus schematic check (LVS): Extraction of a netlist from the polygons of the final layout and automatic compare against the netlist used as input for the layout tool. This ensures the integrity of the layout step.

-

D.2.5.2 Potential safety hazards

– Synthesis, simulation and layout are based on the same cell library (see synthesis for further explanations about common cause failures).

– Faults caused by the layout tool or faults in manual manipulations during layout optimisation are most likely detected by the LVS check.

– The functionality of circuitry created directly at layout level (e.g. analogue blocks, highly area optimised structures) may be extracted from the layout for simulation and verification purposes. Because there is no reference model the layout is based on, faults during the extraction process may falsify simulation results, hiding implementation faults.

– Design rules are dynamic for new process technologies, changing frequent to improve yield and long term stability of the product. Designs based on early design rules may suffer from reliability problems.

D.3 Production

D.3.1 Mask Generation

The structures created on silicon during wafer production are controlled by a set of masks. The mask are generated (drawn) from the layout information (e.g. GDS-II data stream).

D.3.1.1 Verification of the Results

The masks used for production are either manually inspected or automatically compared. Automatic compare requires masks with two identical copies of the layout for each layer.

D.3.1.2 Potential safety hazards

– Manual inspection is error-prone

– Automatic inspection detects only differences between the two copies. Possible common cause faults like faults in the GDS-II data stream or misinterpretation of the layout data are not detected.

-

– Most functional faults are revealed during production test.

D.3.2 Production Test

Test of the final, packaged component using an ASIC tester. Testing may include static power consumption, analogue parameters and selected timing paths. The functionality of the chip is verified using ATPG or functional pattern generated during test insertion.

D.3.2.1 Verification of the Results

Production Test is the final test to ensure that the chip after production is functional equivalent to the netlist used for layout.

D.3.2.2 Potential safety hazards

– Only faults covered by the test pattern set are revealed. Thus, high fault coverage is mandatory.

– Timing is only verified for characteristic paths

-

Appendix E: PLD / FPGA Design Flow

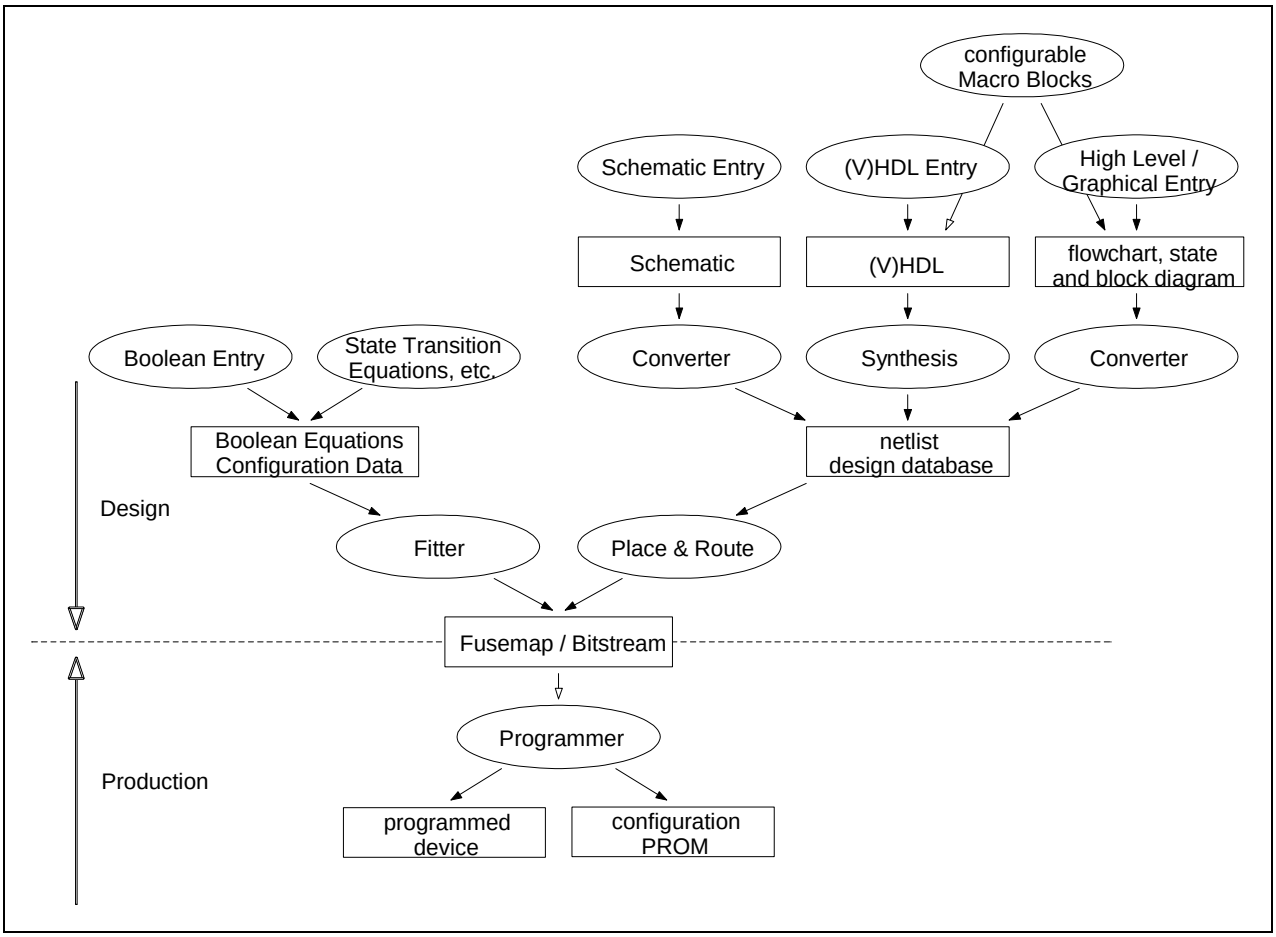


Figure 4: Simplified Design Flow for PLD and FPGA

E.1 Design Entry

E.1.1 Boolean Entry

The simplest type of design description – in most cases used for PLDs only – is to write Boolean Equations (AND-OR product terms). The structure and sequence of operators used in the equation exactly reflect the resources of the PLD (AND-OR matrix). Combinatorial and registered logic is distinguished by special notation, e. g. the operator used for the assignment of the output signal. This type of description is mostly used for simple logic, e. g. address decoding, counters or simple state machines.

-

E.1.1.1 Verification of the Results

Either manually, by walk-through of the equations or with simple simulator tools.

E.1.1.2 Potential safety hazards

- Error prone description, due to very low level of abstraction
- Limited capabilities of the available simulation tools, e. g. to handle feedback-loops
- Common Cause Faults due to build-in simulators
- Tends to be unclear when used for medium and higher complexity

E.1.2 Low Level Hardware Description Languages

In addition to Boolean Equations, low level hardware description languages support constructs for the description of state tables, decision tables and simple arithmetic. Moreover, the design input is less dependent on the actual structure of the target device.

E.1.2.1 Verification of the Results

Either manually or with medium complex build-in simulation tools. Using simulation, it is often possible to specify “stimuli”-“response”-pattern for automated testing.

E.1.2.2 Potential safety hazards

- Low level of abstraction
- Limited capabilities of the available simulation tools
- Common Cause Faults due to build-in simulators

E.1.3 Schematic Entry

See D.1.4

E.1.4 Hardware Description Languages

See D.1.1

-

E.1.5 High Level Design Entry

See D.1.2

E.1.6 Use of Macro Blocks

See D.1.3

E.2 Implementation

E.2.1 Conversion from Schematic to Netlist / Design Database

Translation of the schematic (circuit primitives and interconnections) into a data representation that may be used by the Place & Route tool. The result is either stored in a standard netlist format or a proprietary design database.

E.2.1.1 Verification of the Results

In most cases, no format appropriate for the verification of this intermediate result is provided by the tool vendors.

E.2.1.2 Potential safety hazards

The conversion process may produce a faulty output (resulting in a functional mis-match). The fault may be not revealed at that point in the design flow.

E.2.2 Conversion from High Level Entry to Netlist / Design Database

Basically, as described for the conversion form Schematic to Netlist / Design Database. See E.2.1.

E.2.3 Synthesis

See D.2.1

E.2.4 Device Fitter

Used for PLD / CPLD devices. A device fitter (program) is used to map the input description (e. g. boolean equations) onto the structure of the target device and to

-

create the “fuse map” required for programming. Depending on the complexity of the fitter, the input description needs to be more or less target device orientated.

E.2.4.1 Verification of the Results

Verification of the result is possible in two ways:

- In-circuit, using a device programmed with the generated fuse map or bit stream
- Using simulation. For most simpler devices – small and medium complex PLD – simulation is only supported by the build-in simulators. For more complex devices, additional external (third-party) standard simulators are supported.

E.2.4.2 Potential safety hazards

- In-circuit check of the expected behaviour has a limited fault detection capability, due to the potential problems to stimulate the device and to observe the responses in real-time.
- Build-in (proprietary) simulator tools often have limited capabilities. Moreover the risk for an undetected common cause fault (introduced by the fitter, not revealed by the simulator) increases.
- If third party simulators are supported, the validity of the result is depending on the simulation library. This again is a potential source of a common cause fault.
- For PLD type devices, timing is assumed to be “correct by construction”, so the actual timing is not verified.

E.2.5 Place & Route

Used for FPGA. In a first step, the cells found in the final design database need to be mapped to the cells existing on FPGA. In a second step, the interconnections are routed. In a third step, timing information are derived form the actual placement and interconnection routing and back-annotated for post layout simulation. Finally, the bitstream required for the programming of the device is generated from the placement and interconnection database.

-

E.2.5.1 Verification of the Results

– Simulation of the netlist after layout against the behaviour of the reference model (netlist or RTL source code) with respect to functionality and timing.

– Static timing analysis

– Design rule check (DRC) to guarantee the design rules dictated by the FPGA vendor.

E.2.5.2 Potential safety hazards

– Synthesis, simulation and layout are based on the same cell library (see synthesis for further explanations about common cause failures).

– Fault during bitstream generation.

E.3 Production

Different production schemas are used for volatile (RAM based) and non-volatile (OTP, EEProm or Flash based) devices.

- Volatile devices – typically higher complex FPGAs – need to be re-programmed (loaded) each time after power-on. The information required for this power-on initialisation is usually stored in special non-volatile configuration PROMs; the initialisation is controlled automatically by the FPGA after power-on.
- Non-volatile devices – typically PLDs, CPLDs and lower, up to medium complexity FPGA – are programmed once, prior to the assembly.

E.3.1.1 Verification of the Results

Volatile devices:

- The integrity of the configuration PROMs contents may be checked automatically after programming (readout and compare).
- The information transfer to the volatile component is usually protected by a checksum; this ensures that the devices becomes operational only when a (most likely) correct bit stream is loaded.

Non-volatile devices:

-

- The integrity of the programmed information in a non-volatile device may be checked automatically after programming (readout and compare). In some cases this includes a check of the programmable element (fuse) for correct parameter rating, e. g. “on” or “off” resistance.

E.3.1.2 Potential safety hazards

Volatile devices:

- The protection of the bit stream itself is no guaranty for correct power-on initialisation of the FPGA. Faults may occur when distributing the information in the FPGA (after checksum removal) or stuck-at faults may exist inside the FPGA that result in false behaviour.

Non-volatile devices:

- Only the successful programming may be checked by reading out the programmed pattern. This does not guarantee correct behaviour of the device (same reasoning as for volatile devices).
- Some signal paths in one-time programmable devices may not be checked during chip production, due to the nature of the programmable element. This may lead to unrevealed faults in the device itself.

-

Appendix F: Glossary / Acronyms

ASIC	Application Specific Integrated Circuit
COB	Chip On Board
CPLD	Complex Programmable Logic Device
DRC	Design Rule Check
EEPROM	Electrical Erasable PROM
EPROM	Erasable PROM
FPGA	Field Programmable Gate Array
LVS	Layout versus Schematic Check
MCM	Multi Chip Module
OTP	One Time Programmable (ROM)
PLD	Programmable Logic Device
PROM	Programmable ROM
RAM	Random Access Memory
ROM	Read Only Memory
RTL	Register Transfer Level
VHDL	VSLI (Very Large Scale Integration) Hardware Description Language

-