UG0771
User Guide
Microsemi FPGA Functional Safety

Microsemi
Power Matters.

Microsemi Corporate Headquarters
One Enterprise, Aliso Viejo,
CA 92656 USA
Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Fax: +1 (949) 215-4996
Email: sales.support@microsemi.com
www.microsemi.com

About Microsemi

Microsemi Corporation (Nasdaq: MSCC) offers a comprehensive portfolio of semiconductor and system solutions for aerospace & defense, communications, data center and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs and ASICs; power management products; timing and synchronization devices and precise time solutions, setting the world's standard for time; voice processing devices; RF solutions; discrete components; enterprise storage and communication solutions, security technologies and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Microsemi is headquartered in Aliso Viejo, California, and has approximately 4,800 employees globally. Learn more at www.microsemi.com.

50200771. 1.0 8/17

Contents

Figures

Tables

1 Revision History

The revision history describes the changes that were implemented in the document. The changes are listed by revision, starting with the most current publication.

1.1 Revision 1.0

Revision 1.0 is the first publication of this document.

2 Introduction

Microsemi is committed to providing customers with FPGAs that can be successfully used in critical safety designs. New technology brings new ways for a system containing electronics or programmable electronics to fail. Failures can cause harm to people and property. Even though it is impossible to guarantee a technological system will never fail, it is possible to reduce the risk of failure and to design systems so that if they do fail, they fail safely. To that end, functional safety standards were created, and continue to evolve.

International Electrotechnical Commission (IEC) 61508 Functional Safety of Electrical /Electronic/Programmable Electronic (E/E/PE) Safety-related Systems is a standard created for industrial applications and is intended to be the basis for functional safety in all industries. Microsemi is working with TUV-Rheinland in Cologne, Germany to ensure that the Microsemi SoC FPGA development environment is compliant with IEC 61508.

This document explains how to use Libero® SoC v11.5 SP2 to develop Microsemi's third-generation FPGAs (IGLOO® nano, IGLOO PLUS, ProASIC®3/e, ProASIC3L, and SmartFusion® FPGAs) in compliance with IEC 61508:2010 up to safety integrity level (SIL) 3. Specifically, Libero SoC v11.5 SP2 is qualified for use with the following devices:

- IGLOO_AGL030V2
- IGLOO_AGL030V5
- IGLOO_AGL060V5
- IGLOO_AGL125V5
- IGLOO_AGL250V2
- IGLOO_AGL250V5
- IGLOO_AGL1000V5
- IGLOO_AGLN010V2
- IGLOO_AGLN010V5
- IGLOO_AGLN020V2
- IGLOO_AGLN020V5
- IGLOO_AGLN060V2
- IGLOO_AGLN060V5
- IGLOO_AGLN125V2
- IGLOO_AGLP006V5
- IGLOO_AGLP125V5
- ProASIC3_A3P030
- ProASIC3_A3P060
- ProASIC3_A3P1000
- ProASIC3_A3P125
- ProASIC3_A3P250
- ProASIC3_A3P400
- ProASIC3_A3P600
- ProASIC3_A3P600L
- ProASIC3_A3PE1500
- ProASIC3_A3PE3000
- ProASIC3_A3PE600
- ProASIC3_A3PN010
- ProASIC3_A3PN015
- ProASIC3_A3PN020
- ProASIC3_A3PN030Z
- ProASIC3_A3PN060
- ProASIC3_A3PN060Z
- ProASIC3_A3PN125

- ProASIC3_A3PN125Z
- ProASIC3_A3PN250
- ProASIC3_A3PN250Z
- SmartFusion_ A2F200M3E
- SmartFusion_ A2F500M3G

For a list of soft IP cores that qualified to IEC 61508 SIL3, see Available IP Cores, page 26.

3 Overview of Functional Safety and IEC 61508

IEC 61508 Functional Safety of E/E/PE Safety-related Systems is widely accepted by the industry as the standard to develop and implement safe electronic designs. IEC 61508 provides a generic approach for all safety life cycle activities involved in systems containing E/E/PE elements.

IEC 61508 is an umbrella standard intended to be customized for specific industries. For example, ISO 26262 is the customized IEC 61508-based standard for the automotive industry, and IEC 61513 for the nuclear Industry. If IEC 61508 is not customized for an industry, IEC 61508 itself may be used.

IEC 61508 provides a framework to address the functional safety of systems that contain electrical, electronic, or programmable electronic components. The process described to be compliant with the standard consists of the best practices agreed upon by IEC. The standard consists of the following sections:

1. General Requirements
2. Requirements for E/E/PE Systems
3. Software Requirements
4. Definitions and Abbreviations
5. Examples of Methods for the Determination of Safety Integrity Levels
6. Guidelines for the Application of IEC 61508-2 and IEC 61508-3
7. Overview of Techniques and Measures

Sections 1 through 3 contain the requirements for compliance with IEC 61508. All systems must meet the applicable requirements in sections 1 and 2. In addition, software must meet applicable requirements from section 3.

Understanding there is no such thing as zero risk when it comes to safety, IEC 61508 takes a risk reduction approach. Safety integrity level as defined in IEC 61508-4 is the probability of an E/E/PE safety-related system satisfactorily performing the specified safety functions under all the stated conditions within a stated period of time. SILs 1 through 4 are assigned to the safety functions in a system. The methods to determine the SIL can be found in IEC 61508-5. As the SIL increases, the IEC 61508 requirements become more stringent.

It is not acceptable to consider safety at the component level only. In most situations, safety is achieved by a number of systems, which rely on many technologies (for example, mechanical, hydraulic, pneumatic, electrical, and programmable electronic technologies). Any safety strategy must, therefore, consider not only all the elements within an individual system (for example, sensors, controlling devices, and actuators) but also all the subsystems that make up the overall safety-related system. Although IEC 61508 only addresses E/E/PE systems, it can be used as a framework for other safety-related systems. To be compliant with IEC 61508, the safety of the entire system must be addressed, starting at the beginning of the project. Developers need to consider the following:

- What is the required SIL?
- How is the safety function defined?
- What is the safe state?
- What are the temporal requirements?

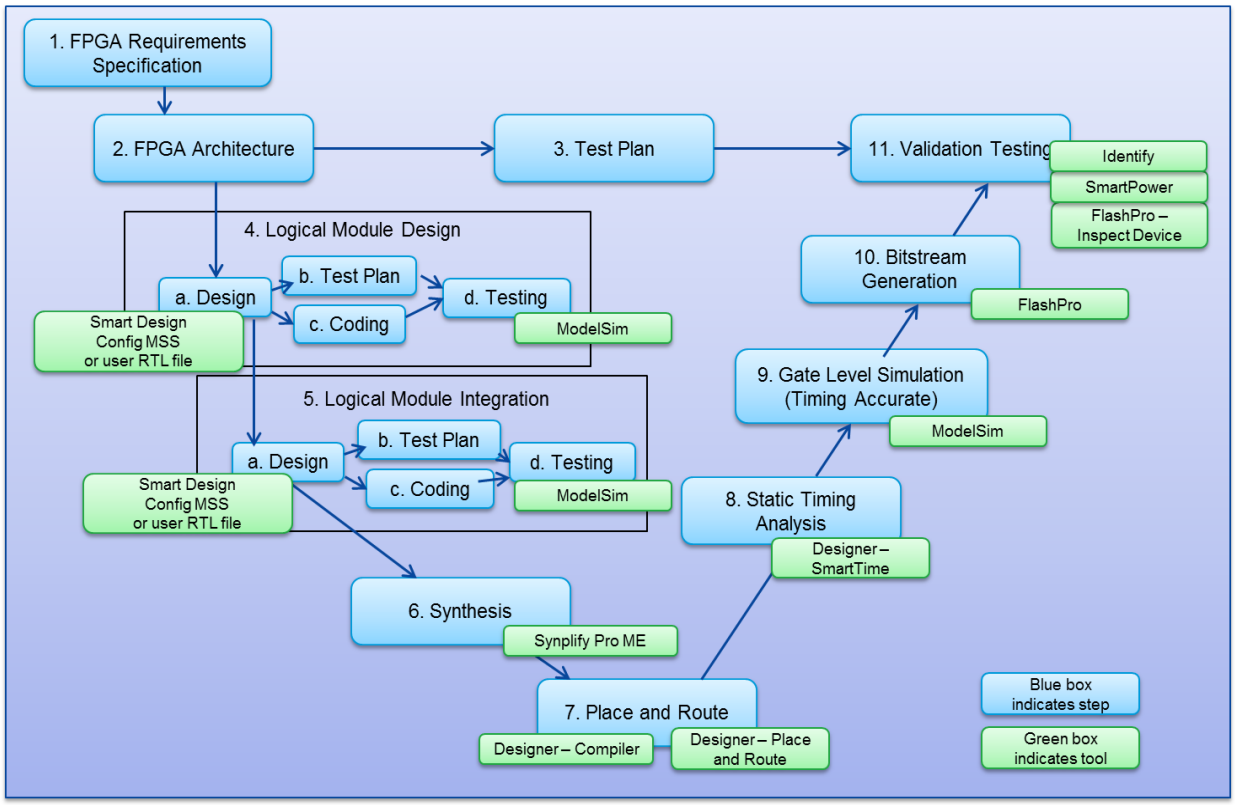
When possible, safety risks are eliminated or reduced in system concept, architecture, or design. Part of compliance involves using best practice processes during development, implementation, and operations. Best practices also need to be used for technical techniques and measures used in FPGA development. The techniques and measures to be used depend on the SIL and are listed in IEC 61508.

IEC 61508 requires the use of a V-model for development. For information about Libero SoC components used in V-Model development, see How to Use Libero SOC v11.5 SP2 in V-Model Development, page 5.

4 How to Use Libero SOC v11.5 SP2 in V-Model Development

Most often, FPGAs are components in a much more complicated system. That puts the FPGA's development near the bottom of the V in the development of the system of interest. Because the development of an FPGA is slightly different from the development of an ASIC, the Microsemi V-model looks slightly different, but it follows the principles and methodology that are shown in Figure 3: ASIC development life cycle (the V-Model) in IEC61508-2. The following figure illustrates how to use Libero SOC v11.5 SP2 in V-Model development.

Figure 1 • **V-Model Development Using Libero SoC**



The following table lists the tools in the Libero SOC tool flow that are used in V-Model development.

Table 1 • **Libero SOC v11.5 SP2 Tools**

Tool	Description	Cat.
SmartDesign	Contains the graphics required for design creation.	T3
Config MSS	Configures in-built microcontroller subsystem components.	T3
Synopsys Synplify Pro ME	Performs high-level optimization before synthesizing the RTL code into specific FPGA logic.	T3

Table 1 • **Libero SOC v11.5 SP2 Tools**

Tool	Description	Cat.
Designer – Compile	Contains a variety of functions that perform legality checks and basic netlist optimization. Checks for netlist errors (bad connections and fan-out problems), removes unused logic (gobbling), and combines functions to reduce logic count and improve performance. Verifies that the device has sufficient resources to fit design.	T3
Designer – Place and Route	Provides the cockpit and engine used to place and route the design.	T3
Designer – SmartTime	Provides gate-level static timing analysis.	T2
Mentor Graphics ModelSim ME	Provides gate-level verification, allowing verification of HDL code line by line.	T2
Designer – Bitstream Generator	Provides the cockpit and engine to export the programing file to program the part, and to export the pin report and IBIS file to design the board.	T3
FlashPro	Programs the FPGA.	T3
Designer – SmartPower	Provides the cockpit and engine to analyze power consumption of the design.	T2
Synopsys Identify ME	Helps find and correct functional design bugs by probing internal signals of the design directly from the programmed FPGA at the system speed.	T2
FlashPro - Inspect Device	Allows users to read analog blocks, eNVM, and µFROM.	T2

The following sections describe each step in V-model development using Libero SoC in more detail.

Note: IEC 61508 also specifies techniques and measures to be used in each step. These are listed in Techniques and Measures (IEC 61508-2, Table F2), page 22, and in IEC 61508–2, table F2.

4.1 Step 1—FPGA Requirement Specification

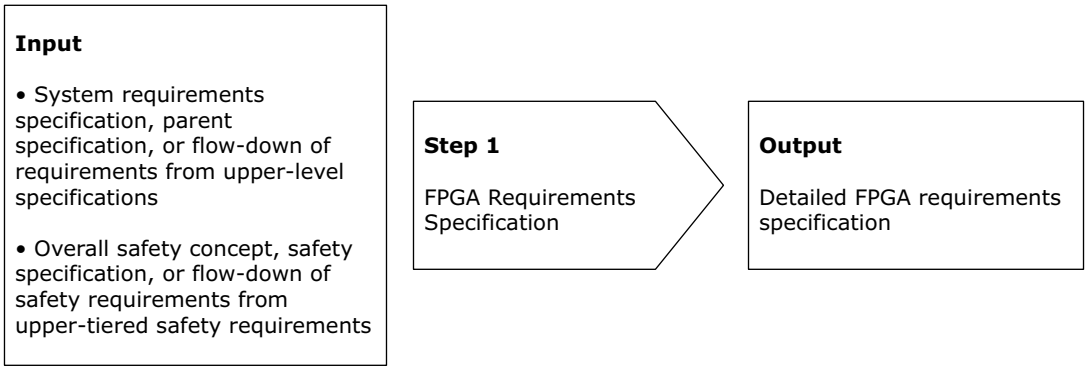
As mentioned in earlier sections, FPGAs are usually part of a bigger system. Requirements for the system, including safety requirements, are defined during early development and then flowed down to the various subsystems. The results of the system requirement flow-down are an input to FPGA development. The first step in the V-model for FPGA development is to create an FPGA specification or requirements document. The specification is written in enough detail for the design to achieve the required safety function.

For the specification to be IEC 61508-compliant, the FPGA specification must be:

- complete and correct in terms of the safety requirements assigned to the FPGA
- free from contradictions
- understandable and unambiguous, and
- capable of being verified and validated.

In addition, the hardware and software constraints must be listed.

Figure 2 • **V-Model Development Step 1—FPGA Requirement Specification**



4.1.1 Microsemi References

None

4.1.2 Verifying Step Was Completed Correctly

- Cross-check detailed FPGA requirements specification against input documents. Using unique identification numbers for the items facilitates the check.
- Peer review the detailed FPGA requirements specification.

4.1.3 Tools

- Standard word-processing software (such as Microsoft Word)
- Requirements authoring and management tools

4.1.4 Specific Techniques and Measures

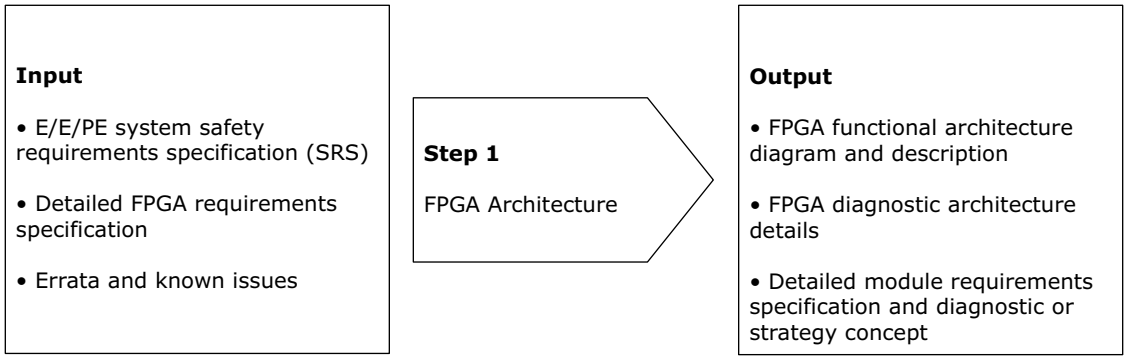
None

4.2 Step 2—FPGA Architecture

In the FPGA Architecture step, the FPGA design is divided into blocks. The requirements in the FPGA specification are partitioned and assigned to the blocks. The interconnections between the blocks and between the blocks and external interfaces are defined. The blocks are assigned such that each block can independently be developed and tested. Typically, a block diagram is created to show the blocks and their interconnections.

As part of creating the FPGA architecture, a specific FPGA may be selected, and IP cores, third-party IP, and standard interfaces may be defined. All necessary safety requirements including design entry, use of specific tools, and sub-modular and modular diagnostic techniques are considered. Any architectural features necessary to check the correct operation of safety design are included. Any known issues with the FPGA devices and tools are considered. The errata for Libero SOC v11.5 SP2 can be found in *Libero SoC v11.5 SP2 Release Notes*.

Figure 3 • **V-Model Development Step 2—FPGA Architecture**



4.2.1 Microsemi References

Errata for Libero SOC v11.5 SP2 in *Libero SoC v11.5 SP2 Release Notes*

4.2.2 Verifying Step Was Completed Correctly

- Cross-check input documents items against output document items.
- Peer review or inspect the architecture.

4.2.3 Tools

- Standard drawing software (such as Microsoft Visio)
- Standard word-processing software (such as Microsoft Word)

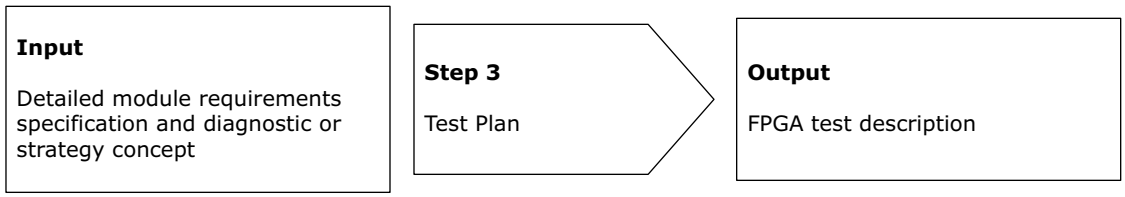
4.2.4 Specific Techniques and Measures

None

4.3 Step 3—Test Plan

In the Test Plan step, a test description or test specification is created for the fully-integrated FPGA. Each point in the FPGA specification or functional requirements document is addressed. Tests are specified to verify correct functionality and possible fault conditions.

Figure 4 • **V-Model Development Step 3—Test Planning**



4.3.1 Microsemi References

None

4.3.2 Verifying Step Was Completed Correctly

- Cross-check testable items in the design document against the numbered tests in the test description
- Peer review the test strategy and coverage

4.3.3 Tools

Standard word-processing software (such as Microsoft Word)

4.3.4 Specific Techniques and Measures

None

4.4 Step 4—Logical Module Design

Logical module design consists of designing individual logical modules, specifying the tests necessary to verify functionality, coding the modules, and then testing them.

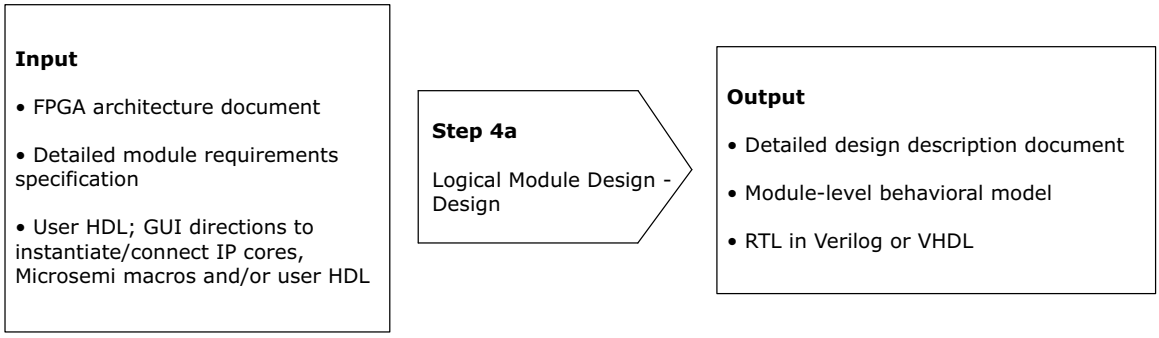
4.4.1 Step 4a—Logical Module Design: Design

The modules or blocks in the FPGA are designed during the Design phase of the Logical Module Design step. Factors considered during this step are RAM usage and arrangement, clocking resources (PLLs) routing and arrangement, block I/O connectivity, and bus types. A design document is created, which details a method to achieve the block's requirements. The design document may be at the level of specifying state machine functions, mathematical functions, and detailed block I/O definitions. It must be of sufficient detail to allow a competent engineer to fully implement the block in the FPGA, including diagnostics.

At this stage, it is recommended that the behavior of the block be modeled. A model can be used to verify FPGA architecture and provide a method for checking the final block implementation. The model may be implemented in a high-level modeling language.

For some designs, it may be desirable to include Microsemi's IP cores. These IP cores are called Microsemi DirectCores and can be found in the IP catalog in Libero SOC v11.5 SP2. Microsemi IP cores qualified for use up to IEC 61508 SIL 3 are listed in Available IP Cores, page 26.

Figure 5 • **V-Model Development Step 4a—Logical Module Design: Design**



4.4.1.1 Microsemi References

Information about SmartDesign, Config MSS, and catalog core generation from *Libero SoC v11.5 User Guide*

4.4.1.2 Verifying Step Was Completed Correctly

- Cross-check the input specification against the output design document.
- Peer review documents.
- Config MSS:
  - Use RTL simulation of the generated design to verify intended functionality.
  - Review reports, warnings, and errors to determine if output is as expected.
  - Check output file dates with respect to the inputs.
  - Use gate-level simulation of the generated design to verify intended functionality.
  - Review timing constraints in SmartTime.
  - Use a lint tool.

- SmartDesign:
  - Use RTL simulation of the generated design to verify intended functionality.
  - Use gate-level simulation of the generated design to verify intended functionality.
  - Review reports, warnings, and errors to determine if output is as expected.
  - Review output logs.
  - Check output file dates with respect to the inputs.
  - Use of a lint tool.

4.4.1.3 Tools

- Standard word-processing software (such as Microsoft Word)
- SmartDesign
- Config MSS

4.4.1.4 Specific Techniques and Measures

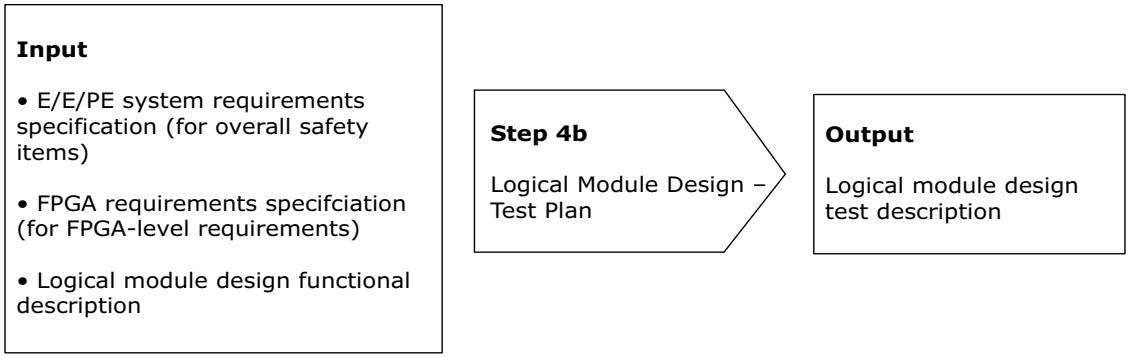
None

4.4.2 Step 4b—Logical Module Design: Test Plan

In the Module Design – Test Plan step, a test description or test specification is created. The test document describes the tests to be used to verify that the block design meets all its requirements.

Each specification item or functional requirement is addressed. Tests are specified to verify correct functionality and possible fault conditions. Tests are also specified to verify the capability of the diagnostic features within the block.

Figure 6 • **V-Model Development Step 4b—Logical Module Design: Test Plan**



4.4.2.1 Microsemi References

Information about creating a SmartDesign test bench and creating an HDL test bench in *Libero SoC v11.5 User Guide*, pages 80–81

4.4.2.2 Verifying Step Was Completed Correctly

- Cross-check testable items in the design document against numbered tests in the test description
- Peer review the test strategy and coverage

4.4.2.3 Tools

- Standard word-processing software (Microsoft Word)
- Libero SoC – Create SmartDesign Testbench
- Libero SoC – Create HDL Testbench

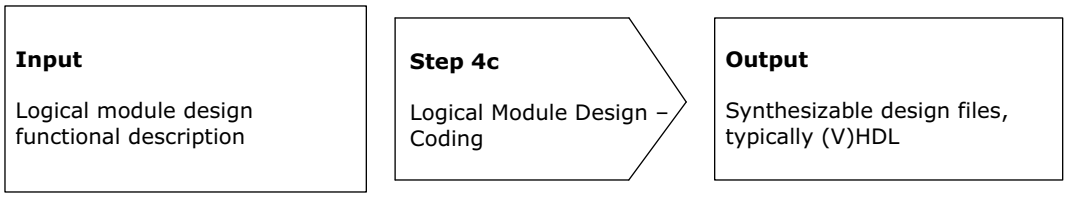
4.4.2.4 Specific Techniques and Measures

None

4.4.3 Step 4c—Logical Module Design: Coding

In the Logical Module Design – Coding step, the detailed block functional description of the block is translated into a synthesizable design description, which typically takes the form of a (V)HDL description of the circuit functions. A standard text editor may be used for design entry. Libero SoC v11.5 SP2 offers options to use SmartDesign, Config MSS, and the Configure catalog to instantiate, configure, and connect cores and peripherals. Structured diagrammatic methods and a design description in (V)HDL are highly recommended in IEC 61508 for all SILs. Schematic entry and Boolean equations are not recommended for SIL 3.

Figure 7 • **V-Model Development Step 4c—Logical Module Design: Coding**



4.4.3.1 Microsemi References

None

4.4.3.2 Verifying Step Was Completed Correctly

- Use a lint tool (if applicable).
- Conduct a code inspection or walkthrough.
- Config MSS:
  - Use RTL simulation of the generated design to verify intended functionality.
  - Review reports, warnings, and errors to determine if output is as expected.
  - Cross-check output file dates against the inputs.
  - Use gate-level simulation of the generated design to verify intended functionality.
  - Review timing constraints in SmartTime.
  - Use a lint tool.
- SmartDesign:
  - Use RTL simulation of the generated design to verify intended functionality.
  - Use gate-level simulation of the generated design to verify intended functionality.
  - Review reports, warnings, and errors to determine if output is as expected.
  - Review output logs.
  - Cross-check output file dates against the inputs.
  - Use a lint tool.

4.4.3.3 Tools

- Standard text editor
- Configure MSS
- SmartDesign

4.4.3.4 Specific Techniques and Measures

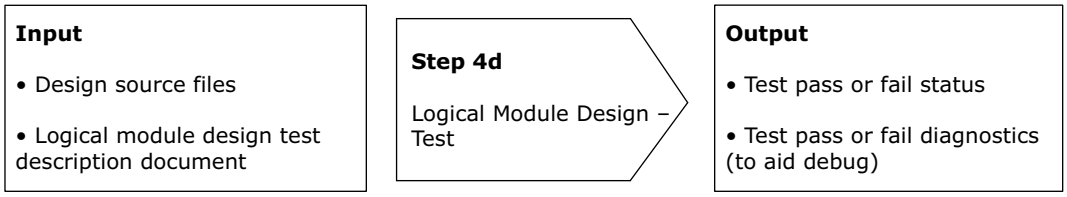
- Structured Description: *IEC 61508-2 Table F.2.1*
- Design Description in (V)HDL: *IEC 61508-2 Table F.2.2*
- Schematic Entry: *IEC 61508-2 Table F.2.3*
- Restricted Use of Asynchronous Constructs: *IEC 61508-2 Table F.2.9*
- Code Inspection or Walkthrough: *IEC 61508-2 Table F.2.15*

4.4.4 Step 4d—Logical Module Design: Test

In the Logical Module Design - Test step, the tests specified in the test description document are executed and the results analyzed. Of the many techniques available for testing blocks, those appropriate for testing the design are selected. First, a test bench or executable code is created for each

item in the block's test description document. Test benches are typically created in (V)HDL or Verilog using a standard text editor. The use of scripts is recommended because scripts not only save time but also allow tests to be run with a high degree of reliability and repeatability. Next, the tests are run and the results collected and analyzed. Information about the pass/fail status of each test must be easily accessible by the engineering team and the management. The source code is modified to correct the faults in the software and is retested.

Figure 8 • **V-Model Development Step 4d—Logical Module Design: Test**



4.4.4.1 Microsemi References

None

4.4.4.2 Verifying Step Was Completed Correctly

- Use tools.
- Peer review test results.
- Manually check for valid simulator output.
- Check for the presence of report file and its time and date stamp.
- Check for time and date stamps of simulation library files.
- Mentor Graphics ModelSim ME:
  - Review logs for correctness.
  - Compare results across RTL and multiple gate-level simulations.
  - Verify that the behavior on the FPGA is as expected.
  - Check output file and simulation library dates against the inputs.
  - Manually check for valid simulator output.
  - Confirm that the correct top is used.
  - Peer review simulation scripts, test benches, and test results.

4.4.4.3 Tools

Mentor Graphics ModelSim ME

4.4.4.4 Specific Techniques and Measures

- Application of Proven-in-Use (V)HDL Simulators: *IEC 61508-2 Table F.2.7*
- Functional Test on Module Level (for example, (V)HDL Test benches): *IEC 61508-2 Table F.2.8*
- Coverage of the Verification Scenarios (Test benches): *IEC 61508-2 Table F.2.12*
- Documentation of Simulation Results: *IEC 61508-2 Table F.2.14*
- Application of Proven-in-Use Libraries/CPLD Technologies: *IEC 61508-2 Table F.2.24*

4.4.5 Step 5—Logical Module Integration

The steps for logical module integration are the same as logical module design. The blocks developed during logical module design are combined so that the whole FPGA's design is defined.

4.4.6 Step 5a—Logical Module Integration: Design

The design effort for logical module integration is very similar to the design effort for logical module design except in module integration, the design of the whole FPGA is considered instead of individual blocks. The FPGA architecture document may be used to describe the interaction among the blocks. For more information, see Step 4a—Logical Module Design: Design, page 9.

4.4.7 Step 5b—Logical Module Integration: Test Plan

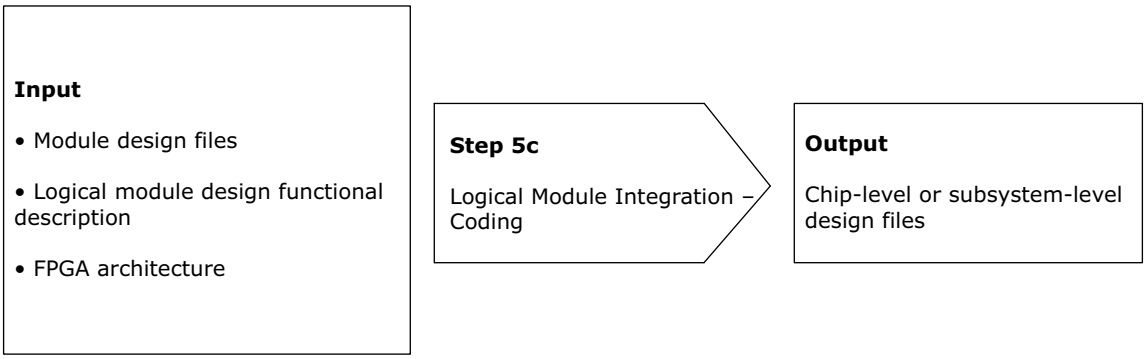
The steps for creating a test plan for logical module integration are the same as those for creating the test plan for logical module design except the testing is done at a higher level in the architecture, usually full-chip testing. For more information, see Step 4b—Logical Module Design: Test Plan, page 10.

4.4.8 Step 5c—Logical Module Integration: Coding

During logical module integration coding, individual blocks developed during the *Logical Module Design* step are combined to create the complete FPGA design.

Libero SmartDesign and Config MSS tools can be used for modular integration. These tools generate code for IP and can simplify code generation and connection of modules. SmartDesign and Config MSS generate an HDL file from a schematic representation. This HDL source can be included in the design in the same way as manually-coded HDL.

Figure 9 • **V-Model Development Step 5c—Logical Module Integration: Coding**



4.4.8.1 Microsemi References

None

4.4.8.2 Verifying Step Was Completed Correctly

- Analyze the report file output for automated steps.
- Check for VHDL source files time and date stamp.
- Config MSS:
  - Use RTL simulation of the generated design to verify intended functionality.
  - Review reports, warnings, and errors to determine if output is as expected.
  - Check output file dates against the inputs.
  - Use gate-level simulation of the generated design to verify intended functionality.
  - Review timing constraints in SmartTime.
  - Use a lint tool.
- SmartDesign:
  - Use RTL simulation of the generated design to verify intended functionality.
  - Use gate-level simulation of the generated design to verify intended functionality.
  - Review reports, warnings, errors to determine if output is as expected.
  - Review output logs.
  - Check output file dates against the inputs.
  - Use a lint tool.

4.4.8.3 Tools

- SmartDesign
- Config MSS
- Libero text editor

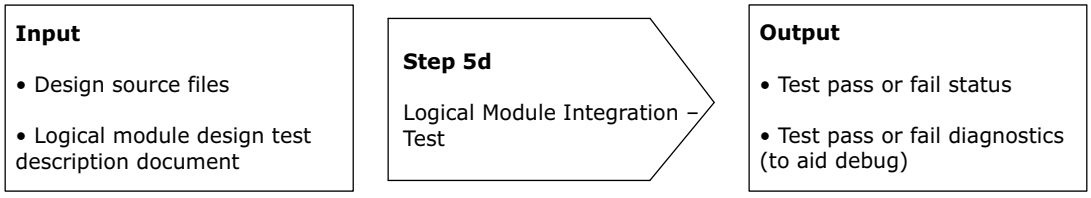
4.4.8.4 Specific Techniques and Measures

- Modularization: *IEC 61508-2 Table F.2.11*
- Application of Validated Soft Cores: *IEC 61508-2 Table F.2.16a*
- Validation of Soft IP Cores: *IEC 61508-2 Table F.2.16b*

4.4.9 Step 5d—Logical Module Integration: Test

The Logical Module Integration - Test step is similar to the Module Design - Test step except instead of individual blocks, the combined blocks as a whole are tested. For more information, see Step 4d—Logical Module Design: Test, page 11.

Figure 10 • **V-Model Development Step 5d—Logical Module Integration: Test**



4.4.9.1 Microsemi References

None

4.4.9.2 Verifying Step Was Completed Correctly

- Peer review or inspect test results.
- Manually check for valid simulator output.
- Check for the presence of the report file and its time and date stamp.
- Check of time and date stamp of simulation library files.
- Mentor Graphics ModelSim ME:
  - Review logs for correctness.
  - Compare results across RTL and multiple gate-level simulations.
  - Verify that the FPGA behavior is as expected.
  - Check output file and simulation library dates against the inputs.
  - Manually check for valid simulator output.
  - Confirm that the correct top is used.
  - Peer review simulation scripts, test benches, and test results.

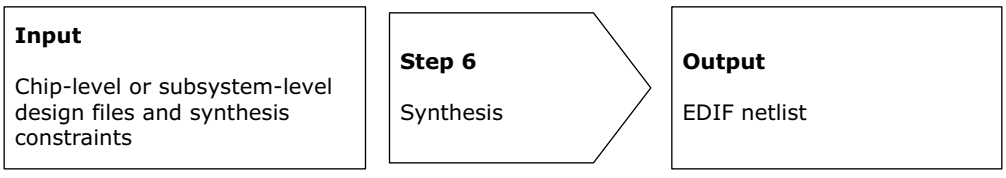
4.4.9.3 Tools

Mentor Graphics ModelSim ME

4.5 Step 6—Synthesis

Synopsys Synplify Pro ME, the synthesis tool in the Libero SOC tool suite, performs high-level optimizations before synthesizing the RTL code into specific FPGA logic.

Figure 11 • **V-Model Development Step 6—Synthesis**



4.5.1 Microsemi References

Synopsys FPGA Synthesis Synplify Pro ME I-2014.03M-SP1 User Guide for Libero SoC v11.5

4.5.2 Verifying Step Was Completed Correctly

- Review generated report files for warnings.
- Check internal project database time and date stamp.
- Check input file list.
- Check output file dates against the inputs.
- Use RTL simulation of the generated design to verify intended functionality.
- Review reports, warnings, and errors to determine if output is as expected.
- Review timing constraints.
- Make sure the Go button does nothing when pushed.

4.5.3 Tools

Synopsys Synplify Pro ME

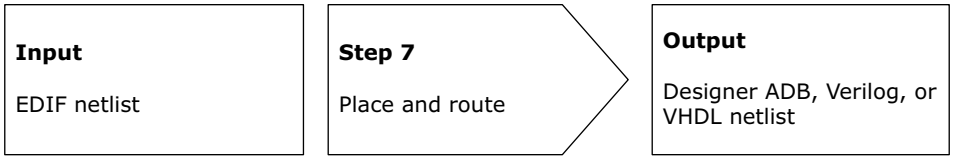
4.5.4 Specific Techniques and Measures

- Internal Consistency Checks: *IEC 61508-2 Table F.2.17*
- Documentation of Synthesis Constraints, Results, and Tools: *IEC 61508-2 Table F.2.22*
- Application of Proven-in-Use Synthesis: *IEC 61508-2 Table F.2.23*
- Application of Proven-in-Use Libraries/CPLD Technologies: *IEC 61508-2 Table F.2.24*
- Script-Based Procedure: *IEC 61508-2 Table F.2.25*

4.6 Step 7—Place and Route

In the Place and Route step, the Designer – Compile subroutine translates the logic functions into a format that can be implemented within the target FPGA. The Designer – Compile subroutine contains a variety of functions that perform legality checking and basic netlist optimization. It checks for netlist errors (bad connections and fanout problems), removes unused logic (gobbling), and combines functions to reduce logic count and improve performance. It also verifies that the FPGA has sufficient resources to meet the design requirements. A netlist is created, which includes the placement and routing of each logic cell. When the default *timing-driven* setting is used, place and route runs automatically.

Figure 12 • **V-Model Development Step 7—Place and Route**



4.6.1 Microsemi References

Information about place and route in *Libero SoC v11.5 User Guide*, pages 108 – 112

4.6.2 Verifying Step Was Completed Correctly

- Analyze tool-generated report files (check for warnings, critical warnings, and so on).
- Check internal project database time and date stamp.
- Check for valid gate-level simulation results.
- Designer – Compile subroutine:
  - Use gate-level simulation of the generated design to verify intended functionality.
  - Make sure the Go button does nothing when pushed.
  - Check output file dates against the inputs.
  - Review reports, warnings, and errors to determine if output is as expected.
  - Check input file list.
  - Manually review constraints.

- Designer – Place and Route subroutine:
  - Use gate-level simulation of the generated design to verify intended functionality.
  - Review reports, warnings, and errors to determine if output is as expected.
  - Check output file dates against the inputs.
  - Make sure the Go button does nothing when pushed.
  - Use SmartTime post layout to verify design timing.
  - Review timing constraints in SmartTime.

4.6.3 Tools

- Designer – Compiler subroutine
- Designer – Place and Route subroutine

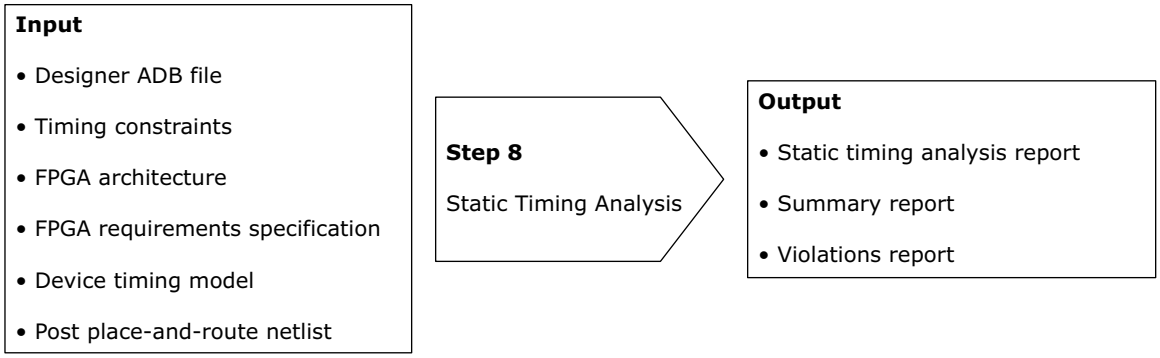
4.6.4 Specific Techniques and Measures

- Justification of Proven-in-Use for Applied Hard Cores: *IEC 61508-2 Table F.2.26a*
- Application of Validated Hard Cores: *IEC 61508-2 Table F.2.26b*

4.7 Step 8—Static Timing Analysis

During static timing analysis, static timing is analyzed at the gate level.

Figure 13 • **V-Model Development Step 8—Static Timing Analysis**



4.7.1 Microsemi References

SmartTime for Libero SoC v11.5 User's Guide

4.7.2 Verifying Step Was Completed Correctly

- Review tool output files for timing failures:
  - Check that the tool reads the correct constraints (.sdc) file.
  - Check the clocks summary report.
  - Check for the presence of the report file and its time and date stamp.
  - Check unconstrained paths in report files.
- Review timing constraints in SmartTime.
- Verify golden simulations against actual behavior on the device.
- Test the device in a production context.
- Review reports, warnings, and errors to determine if output is as expected.
- Check output file dates against the inputs.
- Ensure that the flow is driven to the correct part/speed grade.
- Use SmartTime post layout to verify design timing.
- Confirm that the top specified is the top needed.

4.7.3 Tools

Designer – SmartTime subroutine

4.7.4 Specific Techniques and Measures

- Documentation of Synthesis Constraints, Results, and Tools: *IEC 61508-2 Table F.2.22*
- Static Analysis of the Propagation Delay (STA): *IEC 61508-2 Table F.2.27b*
- Additional Slack (> 20%) For Process Technologies in Use for Less Than 3 Years: *IEC 61508-2 Table F.2.31*

4.8 Step 9—Gate-level Simulation (Timed)

A simulation of the design is created using the netlist from place-and-route. The simulation can first be run without timing data. However, all relevant timing data is later added to the logic simulator to detect any timing violations. It is typical to reuse the simulation test benches created during logical module design testing (see Step 4d—Logical Module Design: Test, page 11).

Figure 14 • **V-Model Development Step 9—Gate-Level Simulation (Timed)**

4.8.1 Microsemi References

Libero SoC v11.5 User Guide, page 112

4.8.2 Verifying Step Was Completed Correctly

- Manually check waveforms.
- Check for time and date stamps in the report file.
- Manually check report file pass or fail status.
- If the simulator gives a false pass to a test, compare simulation results across RTL and multiple gate-level netlists.
- Review logs for correctness.
- Compare results across RTL and multiple gate-level simulations.
- Verify behavior on the FPGA is as expected.
- Check output file and simulation library dates against the inputs.
- Manually check for valid simulator output.
- Confirm that the correct top is used.
- Peer review simulation scripts, test benches, and test results.

4.8.3 Tools

Mentor Graphics ModelSim ME

4.8.4 Specific Techniques and Measures

- Application of Proven-in-Use Libraries/CPLD Technologies: *IEC 61508-2 Table F.2.24*
- Simulation of the Gate Netlist to Check Timing Constraints: *IEC 61508-2 Table F.2.27a*
- Additional Slack (> 20%) for Process Technologies in Use for Less than 3 Years: *IEC 61508-2 Table F.2.31*

4.9 Step 10—Bitstream Generation

The programing files are generated during bitstream generation. The final netlist is used to create the code, which programs the logic cells in the FPGAs. The programing data file (.pdb) is created by the Bitstream Generator subroutine in Designer and sent to the Microsemi FlashPro programmer, which programs the FPGA.

Figure 15 • **V-Model Development Step 10—Bitstream Generation**

4.9.1 Microsemi References

FlashPro for Software v11.5 User's Guide

4.9.2 Verifying Step Was Completed Correctly

- Review tool-generated report files.
- Check programming files time and date stamp.
- Microsemi Designer – Bitstream Generator subroutine:
  - Review reports, warnings, and errors for expected behavior.
  - Verify golden simulations against actual behavior on the device.
  - Check output file dates against inputs.
  - Make sure the Go button does nothing when pushed.
  - Confirm the correct top is used.
  - Check the contents of memories in SmartDebug.
- FlashPro:
  - Review reports, warnings, and errors for expected behavior.
  - Perform functional testing on silicon.
  - Run standalone verify on the programmed device.
  - Verify golden simulations against actual behavior on the device; ensure the root module is set correctly.
  - Make sure the Go button does nothing when pushed.
  - Run device_info in FlashPro for the device being programmed.Tools
- Designer – Bitstream Generator subroutine
- Microsemi FlashPro programmer

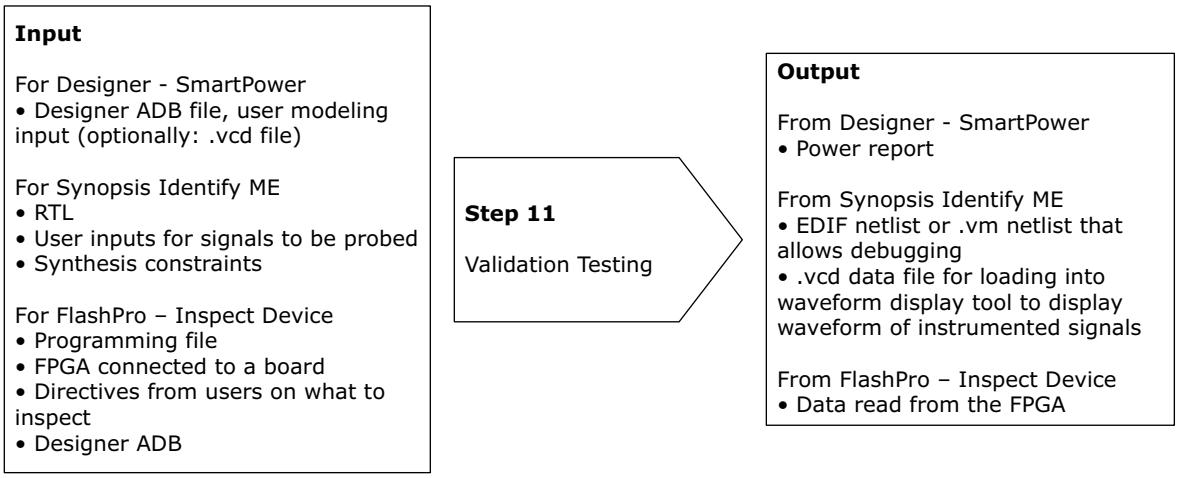
4.9.3 Specific Techniques and Measures

None

4.10 Step 11—Validation Testing

During the Validation Testing step, it is determined whether the FPGA meets the requirements in the FPGA specification. The FPGA is tested according to the test plan (see Step 3—Test Plan, page 8). If the validation is not successful, the Libero SOC tool suite provides tools to help in the debug process. The SmartPower subroutine in Microsemi Designer analyses the power consumption of the design. Synopsys Identify ME helps find and correct functional design bugs by probing internal signals of the design directly from the programmed FPGA at system speed. The Inspect Device subroutine in Microsemi FlashPro allows users to read analog blocks, eNVM, and µFROM.

Figure 16 • **V-Model Development Step 11—Validation Testing**



4.10.1 Microsemi References

- *SmartPower for Libero SoC Software v11.5 User's Guide*
- *Synopsys Identify RTL J-2014.09M-1 Debugger User Guide for Libero SoC v11.5*
- *FlashPro for Software v11.5 User's Guide*

4.10.2 Verifying Step Was Completed Correctly

- Review tool-generated report files.
- To determine if power requirements are correctly calculated, measure power usage on the programmed part in production context.
- To determine the design is correctly modified to trace an internal signal, verify the device behavior is as expected.
- To determine the RTL is in the programming bitstream, prior to production, build the design in a clean environment without Synopsys Identify ME insertion, and check that the checksums generated match the production bitstream.
- To determine that the RTL is in the programming bitstream and the correct signal trace is returned, verify the device behavior is as expected.
- Run standalone verify in FlashPro on the programmed device to determine that the bitstream programmed is the bitstream expected.
- SmartPower:
  - Review power report, warnings, and errors for expected behavior.
  - Test device power usage in a production context.
  - Check output file dates against the inputs.
  - Confirm that the correct top is used.
  - Review input parameters (frequency, probabilities, toggle rates, and operating conditions).
- Synopsys Identify ME:
  - Check the simulation results against the signals on the board.
  - Review reports, warnings, and errors for expected behavior.
  - Check design netlist for instantiation of debugger IP.
  - Check for valid inputs for all points of analysis.
- FlashPro – Inspect Device:
  - Check results against user simulation to ensure the design does not write to an unexpected eNVM address.
  - Check the board or device power supply.
  - Check the cable connection to the JTAG header on board.
  - Check continuity of the JTAG cable.
  - Check the device status.
  - Check for expected results from the device.
  - Check configuration contents from file against those displayed in UI.
  - Check the simulation results against the signals on the board.

- Check that the correct top is set within Libero.
- Check the eNVM Flash Memory Builder configuration.
- Check the UFROM configuration in - Designer, and data assigned.
- Read the device info, and make sure it is correct.
- Read the device status using the View device status option, and check the design versions.
- Review reports, warnings, and errors for expected behavior.

4.10.3 Tools

- Designer – SmartPower
- Synopsys Identify ME
- Microsemi FlashPro

4.10.4 Specific Techniques and Measures

Final Verification and Validation during Mass Production, Per-Unit-Check: *IEC 61508-2 Table F.2.41*

5 Specific Restrictions of Use

Restrictions of use to the Libero SOC tool flow are detailed in *Libero SoC v11.5 SP2 Release Notes*. The Libero SoC v11.5 SP2 certification applies to specific Microsemi devices only. For a list of these devices, see Introduction, page 2.

IP cores referenced in Available IP Cores, page 26 have no restrictions of use.

6 Techniques and Measures (IEC 61508-2, Table F2)

To be compliant with IEC 61508 and to prevent the introduction of faults during the development of an FPGA, it is necessary to use techniques and measures specified in IEC 61508-2, paragraph 7.4.6.7 and table F2. Table 2, page 22 lists the techniques and measures for each SIL (except SIL 4, which is outside the scope of this document). Techniques and measures automatically realized by using the Libero SOC tool flow are checked. The designations HR*, HR, R, -, and NR indicate the importance of the technique or measure. The designations low, medium, and high indicate the effectiveness of the technique or measure.Importance

- **HR*:** The technique or measure is highly recommended for this SIL. No design should exclude this technique or measure.
- **HR:** The technique or measure is highly recommended for this SIL. If this technique or measure is not used, then the rationale behind not using it should be detailed.
- **R:** The technique or measure is recommended for this SIL. If this technique or measure is not used or none of the possible alternatives are used, then the rationale behind not using it should be detailed.
- **–:** The technique or measure has no recommendations for or against being used.
- **NR:** The technique or measure is positively not recommended for this SIL. If this technique or measure is used, then the rationale behind using it should be detailed.

6.0.1 Effectiveness

- **Low:** If used, the technique or measure should be used to the extent necessary to give at least low effectiveness against systematic failures.
- **Medium:** If used, the technique or measure should be used to the extent necessary to give at least medium effectiveness against systematic failures.
- **High:** The technique or measure should be used to the extent necessary to give high effectiveness against systematic failures.

Table 2 • **Techniques and Measures to Prevent Introduction of Faults during FPGA Design and Development (IEC 61508–2, Table F2)**

✓	Ref	Technique/Measure	See IEC 61508-7	SIL 1	SIL 2	SIL 3
Design Entry Phase						
	1	Structured description	E3	HR	HR	HR*
				high	high	high
	2	Design description in (V)HDL	E1	HR	HR	HR*
				high	high	high
	3	Schematic entry	E2	–	–	NR
				high	high	
	4	Design description using Boolean equations		R	R	NR
				high	high	
	5a	For circuit descriptions that use Boolean equations, manual inspection in designs with limited low complexity		HR	HR	HR*
				high	high	high
	5b	For circuit descriptions that use Boolean equations, simulation of state transitions in designs with higher complexity		HR	HR	HR*
				high	high	high

Table 2 • **Techniques and Measures to Prevent Introduction of Faults during FPGA Design and Development (IEC 61508–2, Table F2) *(continued)***

✓	Ref	Technique/Measure	See IEC 61508-7	SIL 1	SIL 2	SIL 3
✓	6	Application of proven-in-use design environment	E4	HR	HR	HR*
				high	high	high
✓	7	Application of proven-in-use design environment (V)HDL simulators	E4	HR	HR	HR*
				high	high	high
	8	Functional test at module level (using, for example, (V)HDL test benches)	E6	HR	HR	HR*
				high	high	high
	9	Restricted use of asynchronous constructs	E9	HR	HR	HR*
				high	high	high
	10	Design for testability (depending on the test coverage in percentage)	E11	R	R	R
				> 95%	> 98%	> 99%
	11	Modularization	E12	R	R	HR
				medium	medium	high
	12	Coverage of the verification scenarios (test benches)	E13	R	R	HR
				medium	medium	high
	13	Observation of coding guidelines	E14	HR	HR	HR*
				high	high	high
	14	Documentation of simulation results	E17	HR	HR	HR
				low	medium	high
	15a	Code inspection	E18	R	R	HR
				medium	high	high
	15b	Walkthrough	E19	R	R	HR
				medium	high	high
	16a	Application of validated soft cores	E20	R	R	HR
				medium	high	high
	16b	Validation of soft cores	E21	R	R	HR
				medium	high	high
Synthesis Phase						
✓	17	Internal consistency checklists (see for example IEC 61508-7, E4)		HR	HR	HR*
				high	high	high
	18a	Simulation of the gate netlist to check timing constraints	E22	R	R	R
				medium	medium	high
	18b	Static analysis of the propagation delay (STA)	E23	R	R	R
				medium	medium	high
	19a	Verification of the gate netlist against a reference model by simulation	E24	R	R	HR
				medium	medium	high

Table 2 • **Techniques and Measures to Prevent Introduction of Faults during FPGA Design and Development (IEC 61508–2, Table F2) *(continued)***

✓	Ref	Technique/Measure	See IEC 61508-7	SIL 1	SIL 2	SIL 3
	19b	Comparison of the gate netlist with the reference model (formal equivalence check)	E25	R	R	HR
				medium	medium	high
	20	For PLD/CPLD in complex designs; check of the design by simulation		R	R	HR
				medium	medium	high
	21	Check of IC vendor requirements and constraints	E26	HR	HR	HR*
				high	high	high
	22	Documentation of synthesis constraints, results, and tools	E27	HR	HR	HR*
				high	high	high
✓	23	Application of proven-in-use synthesis tools	E28	HR	HR	HR*
				high	high	high
✓	24	Application of proven-in-use libraries/CPDL	E29	HR	HR	HR*
				high	high	high
✓	25	Script-based procedures	E30	R	R	HR
				high	high	high
Placement, Routing, Layout Generation Phase						
	26a	Justification of proven-in-use for applied hard cores	E34	HR	HR	HR*
				high	high	high
	26b	Application of validated hard cores	E35	HR	HR	HR*
				high	high	high
	26c	On-line testing of hard cores	E36	HR	HR	HR*
				high	high	high
	27a	Simulation of the gate netlist to check timing constraints	E22	HR	HR	HR*
				high	high	high
	27b	Static analysis of the propagation delay (STA)	E23	HR	HR	HR*
				high	high	high
	28a	Verification of the gate netlist against a reference model by simulation	E24	HR	HR	HR*
				high	high	high
	28b	Comparison to the gate netlist with the reference model (formal equivalence check)	E25	HR	HR	HR*
				high	high	high
✓	29	Design rule check (DRC)	E37	HR	HR	HR
				high	high	high
✓	30	Application of proven-in-use design environments, application of proven-in-use cell libraries	E4	HR*	HR*	HR*
				high	high	high
✓	31	Additional slack (> 20%) for process technologies in use less than three years	E39	HR	HR	HR*
				high	high	high

Table 2 • **Techniques and Measures to Prevent Introduction of Faults during FPGA Design and Development (IEC 61508–2, Table F2) *(continued)***

✓	Ref	Technique/Measure	See IEC 61508-7	SIL 1	SIL 2	SIL 3
Manufacturing Phase						
✓	32	Application of proven-in-use process technology		HR	HR	HR*
				high	high	high
✓	33	Application of proven-in-use device series	E41	HR	HR	HR*
				high	high	high
	34	Proven-in-use manufacturing process	E42	HR	HR	HR
				low	medium	high
	35	Quality control of the manufacturing process	E43	HR	HR	HR
				high	high	high
	36	Manufacturing quality pass of the device	E44	HR	HR	HR
				low	medium	high
	37	Functional quality pass of the device	E45	HR	HR	HR*
				high	high	high
	38	Quality standards	E46	HR	HR	HR
				low	medium	high
✓	39	Quality management, for examples according to ISO 9000		HR	HR	HR
				low	medium	high
	40	Final verification and validation of the FPGA/PLD prototype in the system		HR	HR	HR*
				high	high	high
✓	41	Final verification and validation during mass manufacturing, per-unit-check		R	R	HR*
				high	high	high
	42	Burn-in test	E40	R	R	R
				low	low	medium

Notes:

- Appropriate techniques/measures should be selected according to the safety integrity level. Alternate or equivalent techniques/measures are indicated by a letter following the number. At least one of the alternate or equivalent technique/measures should be applied.
- (V)HDL denotes either very high-speed integrated circuit HDL or Verilog HDL.

7 Available IP Cores

The following IP cores are IEC 61508 qualified up to SIL 3. Be sure to use only the version listed in your critical safety design.

Table 3 • **IEC 61508-Qualified Microsemi IP Cores**

IP	Version
Core AHB2APB3	3.1
CoreABC	3.6
CoreAES128	3.3
CoreAHBLite	5.2
CoreAPB3	4.1
CoreEDAC	2.8
CoreGPIO	3.1
CoreLPC	3.2
CorePWM	4.3

8 Failure Rate, Single Event Upset (SEU) Data

This section describes how to use Microsemi FPGA performance data for the functional safety calculations necessary for IEC 61508 certification.

8.1 Microsemi Reliability Report

Microsemi's extensive reliability report contains the statistical data needed for the IEC 61508 functional safety calculations for Microsemi FPGAs. At the time of this publication, the latest version is *RT0001—Microsemi Corporation SoC Products Reliability Report, revision 15*. The Microsemi Reliability Report can also be found on the Microsemi website: *http://www.microsemi.com/company/quality/reliability*. The report is typically updated annually. The figures in the report may be used as a basis for the functional safety calculations required by IEC 61508. However, contact Microsemi for the most up-to-date figures before submitting your design for assessment.

8.2 Conversion from 60% to 70% Confidence

The Microsemi Reliability Report calculates device failure in time (FIT) rates with a 60% confidence model (FIT60). IEC 61508 requires the use of FIT rates calculated at a 70% confidence level (FIT70).

When no failures are observed for a device, the straightforward conversion from 60% confidence to 70% confidence level shown in the example below can be used. Device failure information can be obtained from the Reliability Summary tables in the Microsemi Reliability Report. If your target device has demonstrated a failure, contact Microsemi for further details of FIT recalculation.

8.3 FIT Formula

Microsemi uses the JESD85 (Methods for Calculating Failure Rates in Units of FITs) standard to calculate FIT rates. The formula for failure rate is:

$$\textit{Failure rate} = X^2/(2 \times \textit{A.F.} \times \textit{device hours})\ \textit{failures/hour}$$

In this formula, the chi-squared term $X^2/2$ gives the probability estimation for the number of failures or rejects. The term A.F. refers to the acceleration factor.

Note: For a full explanation of the method to calculate failure rate, refer to the *Standard FIT Rate and MTTF Assumptions* section in the Microsemi Reliability Report.

8.4 Failure Rate Prediction

When calculating for 60% confidence, the chi-squared value (X^2) is 1.83 based on zero failures. When calculating for 70% confidence, the chi-squared value (X^2) is 2.41 based on zero failures. By inspection, to convert from 60% confidence to 70% confidence, the FIT_{60} rates are multiplied by (2.41/1.83), or 1.32.

Therefore, to use the published Microsemi FIT rates in calculations requiring a 70% confidence level, multiply the 60% confidence figures by a factor of 1.32:

$$\textit{FIT70} = 1.32 \times \textit{FIT60}.$$

Note: This approach is only valid for devices that show no failures. Contact Microsemi for further details of FIT recalculation if your target device has demonstrated a failure.

8.4.1 Example: G3 Products Based on 0.13µm

This example calculation uses FIT information from the Microsemi Reliability Report IGLOO - AGL 0.13µm UMC Flash CMOS FPGA Lifetest Data.

Using chi-squared distribution at 60% confidence level

Acceleration Factor (A.F.) = 77.94

Assuming Ea = 0.7 Ev, Tstress= 125 °C, Tuse= 55 °C

Based on combined G3 product (IGLOO - AGL 0.13µm UMC Flash CMOS FPGA)

Device Hours = 1732396 with zero failures

FIT60 = X^2 × 109/2x (device hrs × A.F) = (1.83 × 109)/(2 × 1732396 × 77.94) = 6.78 FITs

Based on Microsemi conversion, FIT70 = 1.32 × FIT60

FIT at 70% Confidence level = 1.32 × 6.78 = 8.94 FITs

8.5 Adjustment for Alternative Operating Conditions

The Microsemi Reliability Report gives FIT rates for particular device operating conditions such as voltage and temperature. If different operation conditions are experienced, then new acceleration factors for each failure mechanism must be used to calculate the FIT rate. For information about FIT calculation, refer to the *Standard FIT Rate and MTTF Assumptions* section in the Microsemi Reliability Report.

8.6 Soft Error Rate (SER)

Microsemi flash-based FPGAs are immune to SER changes in programmable fabric configuration because of natural radiation in terrestrial, airborne, and space applications. Single-bit errors in the logic fabric flip-flops and in embedded SRAM occur at rates that can be easily managed by established error correction techniques such as shortened hamming codes and forward error correcting codes. The upset rates are shown as FIT rates below. (1 FIT is one failure in 10^9 hours):

Changes to configuration (firm errors) FIT rates per megabit of configuration memory: Immune, no failures detected at sea level at 5,000 ft., 30,000 ft., or 60,000 ft.

Flip-Flop SEU FIT rates: 889 per million flip-flops (note that each FPGA contains significantly fewer than one million flip-flops)

Embedded memory SEU FIT rates: 1,580 per megabit (note that each FPGA contains significantly fewer than 1 megabit of SRAM)

Contact Microsemi for more information.

9 Appendix: Safety Compliance Checklists

To ensure steps are not inadvertently skipped during the development of an IEC 61508-compliant FPGA, it is important to use checklists. The two tables in this section can be used as checklists during FPGA development in addition to Table 2, page 22.

The following table specifies the document structure and naming convention that must be followed in order to meet IEC 61508 requirements.

Table 4 • **Document Checklist for Using Libero SoC v11.5 SP2 in V-Model Development**

V-Model Step		Input Document	✓	Output Document	✓
2	FPGA Architecture	FPGA RequirementsSpecification.doc		PROJECT_DesignDescription.doc	
				PROJECT_synthesis_constraints.doc	
				PROJECT_timing_constraints.doc	
				PROJECT_placement_constraints.doc	
				PROJECT_Timing_Specification.doc	
				PROJECT_Power_Specification.doc	
3	Test Plan	PROJECT_DesignDescription.doc		PROJECT_TestDescription_TopLevel.doc	
		PROJECT_synthesis_constraints.doc			
		PROJECT_placement_constraints.doc			
		PROJECT_timing_constraints.doc			
4	**Logical Module Design**				
4a	Design	PROJECT_DesignDescription.doc		PROJECT_Design_TopLevel_Specification.doc	
				PROJECT_Design_Module-1_Specification.doc	
				PROJECT_Design_Module-2_Specification.doc	
				PROJECT_Design_Module-3_Specification.doc	
				PROJECT_timing_constraints.sdc	
				PROJECT_design_constraints.fdc	
				PROJECT_physical_design_constraints.pdc	
4b	Test Plan	PROJECT_TestDescription_TopLevel.doc		PROJECT_TestDescription_Module-1.doc	
				PROJECT_TestDescription_Module-2.doc	
				PROJECT_TestDescription_Module-3.doc	

Table 4 • **Document Checklist for Using Libero SoC v11.5 SP2 in V-Model Development *(continued)***

V-Model Step	Input Document	✓	Output Document	✓
4c Coding	PROJECT_Design_Module-1_Specification.doc		RTL_Module-1 (v/vhd)	
	PROJECT_Design_Module-2_Specification.doc		RTL_Module-2 (v/vhd)	
	PROJECT_Design_Module-3_Specification.doc		RTL_Module-3 (v/vhd)	
	PROJECT_TestDescription_Module-1.doc		TB_Module-1 (v/vhd)	
	PROJECT_TestDescription_Module-2.doc		TB_Module-2 (v/vhd)	
	PROJECT_TestDescription_Module-3.doc		TB_Module-3 (v/vhd)	
4d Test	PROJECT_TestDescription_Module-1.doc		TB_Module-1.rpt	
	PROJECT_TestDescription_Module-2.doc		TB_Module-2.rpt	
	PROJECT_TestDescription_Module-3.doc		TB_Module-3.rpt	
	RTL_Module-1 (v/vhd)			
	RTL_Module-2 (v/vhd)			
	RTL_Module-3 (v/vhd)			
	TB_Module-1 (v/vhd)			
	TB_Module-2 (v/vhd)			
	TB_Module-3 (v/vhd)			

Table 4 • **Document Checklist for Using Libero SoC v11.5 SP2 in V-Model Development *(continued)***

V-Model Step	Input Document	✓	Output Document	✓
5 Logical Module Integration				
5a Design	RTL_Module-1 (v/vhd)		RTL_Top_Level_SmartDesign (v/vhd)	
	RTL_Module-2 (v/vhd)		RTL_Sub_SmartDesign1 (v/vhd)	
	RTL_Module-3 (v/vhd)		RTL_Sub_SmartDesign2 (v/vhd)	
	PROJECT_Design_TopLevel_Specification.doc		RTL_Core1_file1 (v/vhd)	
	PROJECT_DesignDescription.doc		RTL_Core1_file2 (v/vhd)	
			RTL_Core2_file1 (v/vhd)	
			RTL_Core2_file2 (v/vhd)	
			RTL_MSS_Core_Module (v/vhd)	
			RTL_MSS_Sub_Components_Module (v/vhd)	
			RTL_Firmware-Module.c	
			RTL_Firmware-Module.h	
			MSS_Hardware_Configuration.efc (embedded flash configuration file)	
			TB_MSS.bfm	
			TB_PROJECT.bfm	
			Design_Information.xml	
			DRC_Report.xml	
5b Test Plan	PROJECT_TestDescription_TopLevel.doc		PROJECT_Instrumentation_Specification.doc	
			PROJECT_Inspect_Specification.doc	
5c Coding	PROJECT_TestDescription_TopLevel.doc		TB_Top_Level (v/vhd)	
	PROJECT_Design_TopLevel_Specification.doc		RTL_Top_Level (v/vhd)	

Table 4 • **Document Checklist for Using Libero SoC v11.5 SP2 in V-Model Development *(continued)***

V-Model Step	Input Document	✓	Output Document	✓
5d Test	RTL_Top_Level_SmartDesign (v/vhd)		TB_sim.rpt	
	RTL_Sub_SmartDesign1 (v/vhd)			
	RTL_Sub_SmartDesign2 (v/vhd)			
	RTL_Core1_file1 (v/vhd)			
	RTL_Core1_file2 (v/vhd)			
	RTL_Core2_file1 (v/vhd)			
	RTL_Core2_file2 (v/vhd)			
	RTL_MSS_Core_Module (v/vhd)			
	RTL_MSS_Sub_Components_Module (v/vhd)			
	RTL_Firmware-Module.c			
	RTL_Firmware-Module.h			
	TB_MSS.bfm			
	TB_PROJECT.bfm			
	TB_Top_Level (v/vhd)			

Table 4 • **Document Checklist for Using Libero SoC v11.5 SP2 in V-Model Development *(continued)***

V-Model Step	Input Document	✓	Output Document	✓
6 Synthesis	RTL_Top_Level (v/vhd)		PROJECT_design_synthesis.edn	
	RTL_MSS_Core_Module (v/vhd)		PROJECT_design_synthesis (v/vhd) (generated by rwnetlist.exe)	
	RTL_MSS_Sub_Components_Module (v/vhd)		PROJECT_design_synthesis.srr	
	RTL_RTL_Top_Level_SmartDesign (v/vhd)		PROJECT_design_run_options.txt	
	RTL_Sub_SmartDesign1 (v/vhd)		PROJECT_timing_constraints.sdc (forward annotated)	
	RTL_Sub_SmartDesign2 (v/vhd)			
	RTL_Core1_file1 (v/vhd)			
	RTL_Core1_file2 (v/vhd)			
	RTL_Core2_file1 (v/vhd)			
	RTL_Core2_file2 (v/vhd)			
	PROJECT_timing_constraints.sdc			
	PROJECT_design_constraints.fdc (only through standalone Synplify Pro)			
	PROJECT_synthesis_constraints.doc			
	Designer – Compile Inputs		**Designer – Compile Outputs**	
	PROJECT_design_synthesis.edn		PROJECT_design.adb (internal tool database file)	
	PROJECT_timing_constraints.sdc (forward annotated)		PROJECT_design_compile.rpt	
	PROJECT_physical_design_constraints.pdc		PROJECT_design_netlist (v/vhd/adl/afl)	
7 Place-and-Route	PROJECT_design.adb (internal tool database file)		PROJECT_design.adb (updated adb file with post-layout placement and routing information)	
	PROJECT_placement_constraints.doc		PROJECT_design_place_and_route_report.txt	
			PROJECT_design_iobank_report.txt	
			PROJECT_design_globalnet_report.txt	
			PROJECT_design_globalusage_report.txt	
			PROJECT_design_ba (v/vhd)	
			PROJECT_design_ba.sdf	

Table 4 • **Document Checklist for Using Libero SoC v11.5 SP2 in V-Model Development *(continued)***

V-Model Step		Input Document	✓	Output Document	✓
8	Static Timing Analysis	PROJECT_design.adb (internal tool database file)		Timing.rpt	
		PROJECT_Timing_Specification.doc		Timing_violations.rpt	
				Datasheet.rpt	
				Bottleneck.rpt	
				Constraints_coverage.rpt	
				Combinational_loops.rpt	
9	Gate-Level Simulation (Timed)	PROJECT_design_ba (v/vhd)		PROJECT_design.vcd	
		PROJECT_design_ba.sdf		PROJECT_design_synthesis.wlf	
		TB_Top_Level (v/vhd)		TB_sim.rpt	
		PROJECT_macro_library (v/vhd) (src code or pre-compiled, from libero)			
		PROJECT_TestDescription_TopLevel.doc			
10	Bitstream Generation	PROJECT_design.adb (internal tool database file)		PROJECT_design.pdb	
				PROJECT_design.stp	
		PROJECT_design.fdb (SmartFusion only, input into FlashPro)		PROJECT_design.fdb (for SmartFusion FPGAs only; intermediate output of flow; used as input to FlashPro)	
				PROJECT_design.dat	
				PROJECT_design.svf	
				PROJECT_ieee_1532 (.isc, .bsd)	
				PROJECT_design_secured.pdb	
				PROJECT_design.ibs	
				PROJECT_program.log	

Table 4 • **Document Checklist for Using Libero SoC v11.5 SP2 in V-Model Development *(continued)***

V-Model Step	Input Document	✓	Output Document	✓
11 Validation Testing	PROJECT_design.adb (internal tool database file)		report_power (txt/csv/xml)	
	PROJECT_Power_Specification.doc		report_power_sequencer (txt/csv/xml)	
	RTL_Module-1 (v/vhd)		report_power_peak_analyzer (txt/csv)	
	RTL_Module-2 (v/vhd) RTL_Module-3 (v/vhd)		report_power_activity_map (txt/csv)	
	PROJECT_timing_constraints.sdc		PROJECT_instrumented_design_synthesis.edn	
	PROJECT_Instrumentation_Specification.doc		PROJECT_instrumented_design_synthesis.vm	
	PROJECT_Design_TopLevel_Specification.doc		PROJECT_instrumented_design_synthesis.prj	
	PROJECT_design.pdb or		PROJECT_instrumented_design_synthesis.vcd	
	PROJECT_design.stp		PROJECT_design_HW.rpt	
	PROJECT_Inspect_Specification.doc			

The following table is a checklist of activities required during FPGA development for compliance with IEC 61508.

Table 5 • **Libero SoC v11.5 SP2 V-Model Activities Checklist**

Reference Type (IEC 61508 or Libero SoC-Specific)	Item	✓
Management		
1-6.2.1	Have one or more persons been clearly identified as responsible for functional safety?	
1-6.2.2	Is there a policy and strategy in place for achieving functional safety including communication and evaluation?	
1-6.2.3	Were all organizations and individuals responsible for carrying out activities regarding the safety-related system identified and their responsibilities are communicated to them?	
1-6.2.4	Does a communication plan exist?	
1-6.2.5	Are procedures in place to address safety-related recommendations from: a) hazard and risk analysis b) functional safety assessment c) verification activities d) validation activities e) configuration management f) incident reporting and analysis?	
1-6.2.6	Are procedures in place to ensure all hazardous events are analyzed and recommendations made to minimize the probability of a repeat occurrence?	
1-6.2.7	Are the requirements for periodic safety audits defined including: frequency, level of independence of those carrying out the audits, necessary documentation, and follow-up activities?	

Table 5 • **Libero SoC v11.5 SP2 V-Model Activities Checklist**

Reference Type (IEC 61508 or Libero SoC-Specific)	Item	✓
1-6.2.8	Are procedures in place for initiation and approval of modifications to safety-related systems?	
1-6.2.9	Are procedures in place for maintaining accurate information on hazards, hazardous events, and safety functions?	
1-6.2.10	Is a configuration management system in place and does it include a method to prevent unauthorized items from entering service?	
1-6.2.10	Is a procedure in place to identify all constituent parts of an item?	
1-6.2.11	Is a procedure in place to provide training and information for emergency services where appropriate?	
1-6.2.12	Have all management and technical activities been identified to ensure all functional safety requirements are met?	
1-6.2.12	Have all the measures and techniques necessary to meet all IEC 61508 requirements been defined?	
1-6.2.12	Is a plan in place for the functional safety assessments including the definition of the content and activities?	
1-6.2.12	Are procedures in place for identification and assessment of systematic faults which could jeopardize functional safety? Do the assessments include whether the demand rates and failure rates are in accordance with the assumptions made during the design of the system?	
1-6.2.13 through 1-6.2.15	Is there a competence management system in place that ensures individuals working on the safety-related system have the appropriate knowledge and experience?	
1-6.2.17	Do suppliers meet the appropriate critical safety requirements?	
Documentation		
1-5.2.1	Does the documentation for each life cycle phase contain sufficient information for the next life cycle phase to be successful?	
1-5.2.2	Does the documentation contain sufficient information to manage functional safety?	
1-5.2.3	Does the documentation contain sufficient information to conduct a functional safety assessment?	
1-5.2.4	Is documentation available for items specified by IEC 61508? (See Table 4, page 29.)	
1-5.2.5	Is sufficient documentation available to carry out the duties called out by IEC 61508?	
1-5.2.6	Are the documents accurate and concise?	
1-5.2.6	Are the documents easy to understand by the persons who need to use them?	
1-5.2.6	Do the documents suit the purpose for which they are intended?	
1-5.2.6	Are the documents accessible and maintainable?	
1-5.2.7	Do the documents have titles that indicate the scope of the contents?	
1-5.2.7	Do the documents have some form of index arrangement?	
1-5.2.9	Do the documents have a revision index?	
1-5.2.10	Is it possible to readily identify the revision number in the document?	
1-5.2.10	Are the documents structured in such a way to make it possible to search for relative information?	

Table 5 • **Libero SoC v11.5 SP2 V-Model Activities Checklist**

Reference Type (IEC 61508 or Libero SoC-Specific)	Item	✓
1-5.2.11	Are the documents revised, amended, reviewed, and approved under document control?	
E/E/PE System Safety Requirements Specification and Requirement Flow-Down to FPGA		
1-7.10.2.4	Are the requirements structured and written so that they are: clear, precise, unambiguous, verifiable, testable, maintainable, and feasible?	
1-7.10.2.4	Is each safety function defined with requirements expressed in natural or formal language and/or using logic, sequence, or cause and effect diagrams?	
1-7.10.2.6a	Do the functional safety requirements provide comprehensive detailed requirements sufficient for the design and development of the E/E/PE safety-related system, and are they flowed down properly to the FPGA requirements?	
1-7.10.2.6a	Do the functional safety requirements include the manner in which the E/E/PE safety-related systems are intended to achieve or maintain a safe state for the EUC, and are these requirements flowed down properly to the FPGA requirements?	
1-7.10.2.6a	For each safety function, is it specified whether or not continuous control is required, and for what periods it is needed to achieve and maintain a safe state of the EUC? Are these requirements flowed down properly to the FPGA requirements?	
1-7.10.2.6a	Do the functional safety requirements specify whether each safety function is applicable to E/E/PE safety-related systems operating in low demand, high demand, or continuous modes of operation, and are these requirements flowed down properly to the FPGA requirements?	
1-7.10.2.6b	Do the functional safety requirements specify response time performance (the time within which it is necessary for the safety function to be completed), and are these requirements flowed down properly to the FPGA requirements?	
1-7.10.2.6c	Are the E/E/PE safety-related system and operator interfaces necessary to achieve the required functional safety specified, and are these requirements flowed down properly to the FPGA requirements?	
1-7.10.2.6f	Were all relevant modes of operation of the EUC specified and flowed down properly to the FPGA requirements, including: – preparation for use including setting and adjustment, – start-up, teach, automatic, manual, semi-automatic, steady state of operation, – steady state of non-operation, re-setting, shutdown, maintenance, – reasonably foreseeable abnormal conditions?	
1-7.10.2.6g	Was the failure behavior and the required response in the event of failure defined for the E/E/PE safety-related system (for example alarms, automatic shutdown, and so on)? Were these requirements properly flowed down to FPGA requirements?	
1-7.10.2.7c	Was the required duty cycle and lifetime accounted for in the requirements and considered in the selection of the FPGA?	
1-7.10.2.7e	Were the extremes of all environmental conditions likely to be encountered during the E/E/PE system safety life cycle specified, including manufacture, storage, transport, testing, installation, commissioning, operation, and maintenance? Were these requirements properly flowed down to FPGA requirements?	
1-7.10.2.7f	Were the electromagnetic immunity limits that are required to achieve functional safety considered in the derivation of requirements?	
7-C.2.11	Are the requirements traceable forward and backward?	

Table 5 • **Libero SoC v11.5 SP2 V-Model Activities Checklist**

Reference Type (IEC 61508 or Libero SoC-Specific)	Item	✓
Design Entry		
7-E1	Is the functional description at a high abstraction level in a hardware description language, for example, in (V)HDL?	
7-E3	Was a structured or modular approach used in describing the functionality of the circuit so that it is easily readable and understood?	
	Was there a procedural crosscheck of detailed FPGA requirements specification against input documents?	
7-E4	Was a proven-in-use design environment used?	
7-E6	Were functional tests at the module level using test benches completed and documented?	
7-E9	Were asynchronous constructs restricted?	
7-E11	Were structures that are not testable or poorly testable avoided, for example, asynchronous constructs, latches, and on-chip tristate signals, and wired-AND/wired-OR logic and redundant logic?	
7-E12	Was a modular description of the circuit functions with clearly defined interfaces used?	
7-E13	Were the applied verification scenarios quantitative and qualitatively documented?	
7-E14	Were coding guidelines that resulted in a syntactic, semantic, easy-to-read circuit code used?	
7-E17	Was all the data needed for a successful simulation in order to verify the circuit's functions documented?	
7-E18	Was there a review of the circuit description? Did the review include: – checking the code style – verifying the functionality against the specification – checking for defensive coding, error and exception handling?	
7-E19	Was a code walkthrough conducted and documented for a set of test cases?	
7-E20 and E21	Were all soft cores validated?	
Libero SOC V-model	If Config MSS was used, was a RTL simulation of the generated design used to verify intended functionality?	
Libero SOC V-model	If Config MSS was used, were reports, warnings, and errors reviewed to determine if the output was as expected?	
Libero SOC V-model	If Config MSS was used, were the output file dates checked against inputs?	
Libero SOC V-model	If Config MSS was used, was a gate-level simulation of the generated design used to verify intended functionality?	
Libero SOC V-model	If Config MSS was used, were constraints in SmartTime reviewed?	
Libero SOC V-model	If Config MSS was used, was a lint tool used, if appropriate?	
Libero SOC V-model	If SmartDesign was used, was an RTL simulation of the generated design used to verify intended functionality?	
Libero SOC V-model	If SmartDesign was used, were reports, warnings, and errors reviewed to determine if the output was as expected?	
Libero SOC V-model	If SmartDesign was used, were the output file dates checked against inputs?	

Table 5 • **Libero SoC v11.5 SP2 V-Model Activities Checklist**

Reference Type (IEC 61508 or Libero SoC-Specific)	Item	✓
Libero SOC V-model	If SmartDesign was used, was a gate-level simulation of the generated design used to verify intended functionality?	
Libero SOC V-model	If SmartDesign was used, were output logs reviewed?	
Libero SOC V-model	If SmartDesign was used, was a lint tool used, if appropriate?	
Testing		
7-E6	Were functional tests at the module level using test benches completed and documented?	
7-E7	Were functional tests at the top or FPGA level completed and documented?	
Libero SOC V-model	If Mentor Graphics ModelSim ME was used, were logs reviewed for correctness?	
Libero SOC V-model	If Mentor Graphics ModelSim ME was used, were results compared across multiple gate-level simulations?	
Libero SOC V-model	If Mentor Graphics ModelSim ME was used, was behavior at the FPGA level as expected?	
Libero SOC V-model	If Mentor Graphics ModelSim ME was used, were output files and simulation library dates checked against inputs?	
Libero SOC V-model	If Mentor Graphics ModelSim ME was used, was a manual check for valid simulator output performed?	
Libero SOC V-model	If Mentor Graphics ModelSim ME was used, was it confirmed the correct top was used?	
Libero SOC V-model	If Mentor Graphics ModelSim ME was used, was a peer review or inspection conducted of the simulation scripts, test benches, and test results?	
Libero SOC V-model	Was a peer review or inspection of the test strategy and coverage conducted?	
Libero SOC V-model	Was a crosscheck completed to determine if all the testable items in the requirements documents are in the test plan?	
Synthesis		
7-E22	Was the gate netlist simulated to check timing constraints including best- and worst-case conditions at the maximum specified clock rate?	
7-E23	Was a static timing analysis completed and documented?	
7-E24	Was the gate netlist verification against the reference model by simulation completed and documented?	
7-E25	Was the gate netlist compared to the reference model to ensure a functional equivalence check independent of simulation?	
7-E26	Were vendor requirements and constraints checked?	
7-E27	Were all constraints required for an optimal synthesis to generate the final gate netlist documented?	
7-E28	Was a proven-in-use synthesis tool been used?	
7-E29	Were only proven-in-use libraries used?	
7-E30	Were script-based procedures used in the synthesis cycles?	
Libero SOC V-model	Were the report files generated by Synopsys Synplify Pro ME reviewed for warnings?	
Libero SOC V-model	Was the internal project database time and date stamp in Synplify Pro ME checked?	
Libero SOC V-model	Was the input file list to Synopsys Synplify Pro ME checked?	

Table 5 • **Libero SoC v11.5 SP2 V-Model Activities Checklist**

Reference Type (IEC 61508 or Libero SoC-Specific)	Item	✓
Libero SOC V-model	Were the output file dates in Synopsys Synplify Pro ME checked against inputs?	
Libero SOC V-model	Was RTL simulation of the generated design performed to verify intended functionality?	
Libero SOC V-model	Were the Synopsys Synplify Pro ME reports, warnings, and errors checked to determine if the output was as expected?	
Libero SOC V-model	Were the timing constraints checked?	
Placement, Routing, and Layout Generation		
7-E34 and E35	Were only proven-in-use hard cores used?	
7-E36	Was on-line testing performed and documented for hard cores?	
7-E22	Was the gate netlist simulated to check timing constraints including best- and worst-case conditions at the maximum specified clock rate?	
7-E23	Was a static timing analysis completed and documented?	
7-E24	Was the gate netlist verification against the reference model by simulation completed and documented?	
7-E25	Was the gate netlist compared to the reference model so that there was a functional equivalence check independent of simulation?	
7-E37	Was a detailed design rule check (DRC) completed and documented (that is, were the vendor design rules verified and documented, for example, minimum and maximum wire lengths, placement of layout structures, and so on)?	
7-E4	Was a proven-in-use design environment used?	
7-E39	If the process technology was in use less than three years, was at least 20% slack added to the minimum and maximum timing constraints?	
Libero SOC V-model	During static timing analysis, did the tool read the correct constraints (.sdc) file?	
Libero SOC V-model	During static timing analysis, was the clocks summary report reviewed?	
Libero SOC V-model	During static timing analysis, was a check done for the presence of the report file and its time and date stamp?	
Libero SOC V-model	During static timing analysis, were the unconstrained paths in report files checked?	
Libero SOC V-model	During static timing analysis, was the device tested in a production context?	
Libero SOC V-model	During static timing analysis, were reports, warnings, and errors reviewed to determine if the output was as expected?	
Libero SOC V-model	During static timing analysis, were output file dates checked against inputs?	
Libero SOC V-model	During static timing analysis, was it checked that the flow was driven to the correct part/speed grade?	
Libero SOC V-model	During gate-level simulation (timed), was there a peer review or inspection of simulation scripts, test benches, and test results?	
Libero SOC V-model	During gate-level simulation (timed), was there a manual check of waveforms?	
Libero SOC V-model	During gate-level simulation (timed), was there a check for time and date stamps in report file?	
Libero SOC V-model	During gate-level simulation (timed), was there a manual check of the report file pass or fail status?	

Table 5 • **Libero SoC v11.5 SP2 V-Model Activities Checklist**

Reference Type (IEC 61508 or Libero SoC-Specific)	Item	✓
Libero SOC V-model	During gate-level simulation (timed), if the simulator gave a false pass to a test, were the simulation results compared across RTL and multiple gate-level netlists?	
Libero SOC V-model	During gate-level simulation (timed), were the logs reviewed for correctness?	
Libero SOC V-model	During gate-level simulation (timed), were the results compared across RTL and multiple gate level simulations?	
Libero SOC V-model	During gate-level simulation (timed), was it verified expected behavior was happening on the FPGA?	
Libero SOC V-model	During gate level simulation (timed), were the output files and simulation library dates checked against inputs?	
Libero SOC V-model	During gate level simulation (timed), was a manual check done for valid simulator output?	
Libero SOC V-model	During gate level simulation (timed), was it confirmed that the correct top was used?	
Libero SOC V-model	During bitstream generation, were the tool-generated report files reviewed?	
Libero SOC V-model	During bitstream generation, were golden simulations used to determine if hardware was performing as expected?	
Libero SOC V-model	During bitstream generation, were the programming files time and date stamp checked?	
Libero SOC V-model	During bitstream generation, were the reports, warnings, and errors checked for expected behavior?	
Libero SOC V-model	During bitstream generation, were the golden simulations verified against actual behavior on the device?	
Libero SOC V-model	During bitstream generation, were the output file dates checked against inputs?	
Libero SOC V-model	During bitstream generation, was it confirmed that the correct top was used?	
Libero SOC V-model	During bitstream generation, were the contents of memories in SmartDebug checked?	
Libero SOC V-model	Was SmartTime used post layout to verify design timing?	
Libero SOC V-model	Were timing constraints reviewed in SmartTime?	
Final Validation		
Libero SOC V-model	Were the tool-generated report files reviewed?	
Libero SOC V-model	Was the power usage on the programed part in the production context measured to determine if power requirements were correctly calculated?	
Libero SOC V-model	To determine the design was correctly modified, was an internal signal traced to verify expected behavior?	
Libero SOC V-model	To determine that the RTL was in the programming bitstream, prior to production, was the design built in a clean environment without identify insertion and the checksums checked to determine they match the production bitstream?	
Libero SOC V-model	Was the expected behavior verified on the device to determine that the RTL was in the programming bitstream and that the correct signal trace was returned?	
Libero SOC V-model	Was the standalone verify in FlashPro on the programmed device run to determine that the bitstream programmed was the bitstream expected?	
Libero SOC V-model	If SmartPower was used, were the power report, warnings, and errors reviewed for expected behavior?	

Table 5 • **Libero SoC v11.5 SP2 V-Model Activities Checklist**

Reference Type (IEC 61508 or Libero SoC-Specific)	Item	✓
Libero SOC V-model	If SmartPower was used, was the device power usage tested in a production context?	
Libero SOC V-model	If SmartPower was used, were the output file dates checked with respect to the inputs?	
Libero SOC V-model	If SmartPower was used, was it confirmed the correct top was used?	
Libero SOC V-model	If SmartPower was used, were the input parameters (frequency, probabilities, toggle rates, and operating conditions) reviewed?	
Libero SOC V-model	If Synopsys Identify ME was used, were the simulation results checked against the signals on the board?	
Libero SOC V-model	If Synopsys Identify ME was used, were reports, warnings, and errors reviewed for expected behavior?	
Libero SOC V-model	If Synopsys Identify ME was used, was the design netlist checked for instantiation of debugger IP?	
Libero SOC V-model	If Synopsys Identify ME was used, were inputs valid for all points of analysis?	
Libero SOC V-model	If Synopsys Identify ME was used, were reports, warnings, and errors reviewed for expected behavior?	
Libero SOC V-model	If Synopsys Identify ME was used, were functional tests performed and passed in silicon?	
Libero SOC V-model	If Synopsys Identify ME was used, was standalone verify run on the programmed device?	
Libero SOC V-model	If Synopsys Identify ME was used, were the golden simulations verified against actual behavior on the device to ensure that the root module was set correctly?	
Libero SOC V-model	If Synopsys Identify ME was used, was device_info run in FlashPro for the device being programmed?	
Manufacturing		
	Was a proven-in-use technology used in the manufacturing process?	
7-E41	Does the manufacturer of the safety design have sufficient application experience with programmable devices and development tools?	
7-E42	Does the manufacturer have sufficient series production experience?	
7-E43	Does manufacturer have a quality control process that ensures continuous process control?	
7-E44	Is there proof the manufacturer performs selected part stress tests, for example, temperature-humidity bias test of change of temperature tests?	
7-E45	Is there proof that the manufacturer tested the devices for functionality?	
7-E46	Is the manufacturer's quality management system qualified against a quality standard such as ISO 9000?	
	Was there a final verification and validation of the FPGA/PLD prototype in the system?	
	Was there a final verification and validation during mass manufacturing, per-unit-check?	
7-E40	Did the manufacturer perform burn-in tests?	