Design Compiler®
Design Compiler NXT
Design Compiler Graphical
DC Ultra™
HDL Compiler™
Power Compiler™

Functional Safety Manual

November 2021, Revision 2.0

Synopsys, Inc.
690 E. Middlefield Road
Mountain View, CA, 94043
www.synopsys.com

Document control

Revision history

Version	Description	Date
0.9	Initial draft	05-Oct-2017
1.0	First release of the document submitted for review.	10-Jan-2018
1.1	Added revision history, fixed template issues.	06-Feb-2018
1.2	Incorporated review comments from exida review. Fixed boilerplate changes from general feedback.	01-Mar-2018
1.3	Updated CoU-004 and typos with some AoUs; alphabetized Terms table	02-Mar-2018
1.4	Updated content based on certification review	09-Mar-2018
1.5	Update content for Design Compiler NXT	21-Aug-2019
1.6	Update for tool flow diagram and replaced PrimeTime PX with PrimePower	08-Oct-2019
1.7	Updates to Use Case 4	02-Dec-2019
2.0	Updated for ISO 26262 re-certification	30-Nov-2021

Contents

1 Customer Support

This section describes the customer support that is available through the Synopsys SolvNetPlus® customer support website or by contacting the Synopsys support center.

Accessing SolvNetPlus

The SolvNetPlus support site includes an electronic knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. The site also gives you access to a wide range of Synopsys online services, which include downloading software, viewing documentation, and entering a call to the Support Center.

To access the SolvNetPlus site:

1. Go to the web page at https://solvnetplus.synopsys.com/ .
2. If prompted, enter your user name and password. (If you do not have a Synopsys user name and password, follow the instructions to register.)

If you need help using the site, click **Help** on the menu bar.

Contacting Synopsys Support

If you have problems, questions, or suggestions, you can contact the Synopsys support center in the following ways:

- Go to the Synopsys Global Support Centers site on synopsys.com. There you can find e-mail addresses and telephone numbers for Synopsys support centers throughout the world.
- Go to either the Synopsys SolvNetPlus site or the Synopsys Global Support Centers site and open a case online (Synopsys user name and password required).

2 Scope of This Document

This section describes the scope of this document and defines terms used in this document.

Using This Document

The *Design Compiler Functional Safety Manual* describes the proper use of the Design Compiler tool in safety-related applications according to the ISO 26262 standard, and is intended to confirm the compliance of the Design Compiler tool to the standard when used in the context of a tool chain.

The entire family of Synopsys synthesis tools comprising Design Compiler®, Design Compiler NXT, Design Compiler Graphical, DC Ultra™, HDL Compiler™ and Power Compiler™ are covered by this document; but, henceforth all of these products will primarily be referenced in this document as Design Compiler. Section 6 specifically lists out which of the synthesis products apply to that particular use case.

The Design Compiler tool enables the user to take a functional description of their design written in RTL (SystemVerilog or VHDL), translate it to a logically equivalent gate level description of the design, and optimize it for a variety of user provided constraints.

Section 3 describes an overview of the ISO 26262-8, clause 11 and the approach adopted by Synopsys to comply with the requirements of the standard. Section 4 defines the general information such as where to find the latest documentation and installation requirements regarding the use of the Design Compiler tool as a software tool in the development of safety-related applications. Section 5 shows the high-level overview of the tool chain that this product belongs to. Section 6 details the safety-related requirements for safety-qualified use cases of the Design Compiler tool. Section 7 lists the known limitations of the use cases.

Specific documentation for performing design and analysis as part of an ISO 26262 compliant flow is provided in Section 3, Section 5, Section 6 , Appendix A, and Appendix B of this document, the *Design Compiler Functional Safety Manual*.

Terms and Definitions

Term	Definition
alib	A technology library optimized format used by Design Compiler for runtime improvement
AoU	Assumption of Use.

	An action that is assumed and required to be taken by the user of a software tool.
ASIL	Automotive Safety Integrity Level. This is a risk classification scheme defined by the standard ISO 26262. The standard identifies four levels: ASIL A, ASIL B, ASIL C, and ASIL D. ASIL D dictates the highest integrity requirements on a product and ASIL A dictates the lowest.
Component	A part of an electronic system that implements a function in a vehicle. See also Part 1 of the standard ISO 26262 for the definition. The standard also refers to elements and items, but for the *Design Compiler Functional Safety Manual*, there is no difference.
CoU	Condition of Use. A condition of the design, software tool, design environment, or situation that is assumed and required to be fulfilled by the user.
db	A binary file format for storing library and design data
DC	Design Compiler
DCLS	Dual Core Lock Step A safety mechanism that can be implemented in functional safety that have the processors operating in parallel (e.g. in lock step). It is used for error detection of the cores of a single event upset (such as a soft error).
ddc	A binary file format for storing design data
Defect	Product nonconformance.
DMR	Dual Modular Redundancy This is a safety mechanism that can be implemented for functional safety. In most cases, it is used in the context of safety registers for error detection of a single event upset (such as a soft error).
Error	An error is a discrepancy between the actual and the specified or theoretically correct operation of an element. The root causes of an error can be manifold. In this document, the focus is on errors that are introduced or left undetected in a design, due to the malfunction in a software tool (for example, generation of bad logic by a logic synthesis tool, failure of a static timing analysis tool to detect a timing violation).
Fault	An abnormal condition that can cause an element or item to fail.

Fault analysis	An analysis that determines the behavior of a system when a fault is introduced.
FFSM	Failsafe Finite State Machine State machine encoding that is used as a safety mechanism.
FMEA	Failure Mode and Effects Analysis. An analysis that looks at different parts of a system, identifies ways the parts could fail, and determines the causes and effects of these potential failures.
FuSa	Functional Safety
HDL	Hardware Description Language Language for describing designs at RTL. Supports SystemVerilog and VHDL.
Milkyway	A binary file format for storing library and design data
NDM	A binary file format for storing library and design data
RTL	The Register Transfer Level A level of design abstraction where data is moved from register to register
SAIF	Switching Annotation Interchange Format A text syntex for passing switching information between tools.
SDC	Synopsys Design Constraints A TCL based syntax for describing design constraints in Design Compiler
SEU	Single Event Upset Transient faults that can occur in designs, such as impact from ion or electromagnetic radiation
Software / software tool	The Design Compiler tool.
Software tool criteria evaluation	Analysis according to ISO 26262 to determine the required TCL of a software tool.
Software tool qualification	Means to create evidence, that a software tool with low or medium TCL is suitable to be used in the development of safety related products according to ISO 26262.
SolvNetPlus	Synopsys customer support site.

SSF	Safety Specification Format
Standard	In this document, refers to *ISO 26262 Road Vehicles – Functional Safety*, 2011 and 2018 versions.
STAR	Synopsys Technical Action Request. A STAR documents and tracks a product Bug or Enhancement request (called a B-STAR or an E-STAR, respectively). It is stored in the Synopsys internal defect database. Only Synopsys employees can access the internal defect database. However, limited STAR information is available from SolvNetPlus for customers who are associated with the user site of a STAR. Customer contacts are notified automatically when a STAR is filed or when its status changes.
SVF	Synopsys Verification File This file is used as guidance for the Formality product.
TCL	Tool confidence level, as defined by ISO 26262-8, clause 11. Note: The TCL of a software tool does not necessarily indicate whether the tool may malfunction or not. The TCL defines the confidence level that an error in the safety-related design, which is introduced or left undetected by the software tool, can be prevented or detected in subsequent steps of the development flow, before the erroneous safety-related design is released.
TD	Tool error detection, as defined in ISO 26262-8, clause 11.
TI	Tool impact, as defined in ISO 26262-8, clause 11.
TMR	Triple Modular Redundancy This is a safety mechanism that can be implemented for functional safety. In most cases, it is used in the context of safety registers for error correction of a single event upset (such as a soft error).
UPF	Unified Power Format A TCL based syntax for describing low power constraints and design information. Supported by Design Compiler
Use case	A use case is a specific way of using a software tool, that can be characterized by: - a limited set of tool functions and features that are used; - a set of restrictions and constraints that are regarded while using the tool; and

	- a specific goal to be achieved or output to be generated by using the software tool Use cases may be associated with different steps or phases in the design process, or they may describe alternative ways of using the tool for a specific design step.

3
Confidence in the Use of Software Tools According to ISO 26262-8, Clause 11

This section provides an overview of the ISO 26262-8, clause 11. It then describes the approach adopted by Synopsys to comply with the requirements of the standard, and how this is mapped to activities performed by Synopsys and the end user of the Synopsys tools.

Overview of ISO 26262-8, Clause 11

Synopsys EDA software tools contribute significantly to the design specification, implementation, integration, verification and validation of electrical and electronic (E/E) systems and components. If these E/E systems and components are used as part of a safety-related automotive product, an error in these systems or components could have severe consequences on functional safety. Such an error may arise as a result of unforeseen operating conditions or due to a fault introduced during product development, which in turn may be caused by a software tool malfunction. ISO 26262-8, clause 11 (Confidence in the Use of Software Tools) addresses this issue and specifies requirements and methods which aim to minimize the risk of faults in the developed product due to malfunctions of a software tool affecting the product's functional safety.

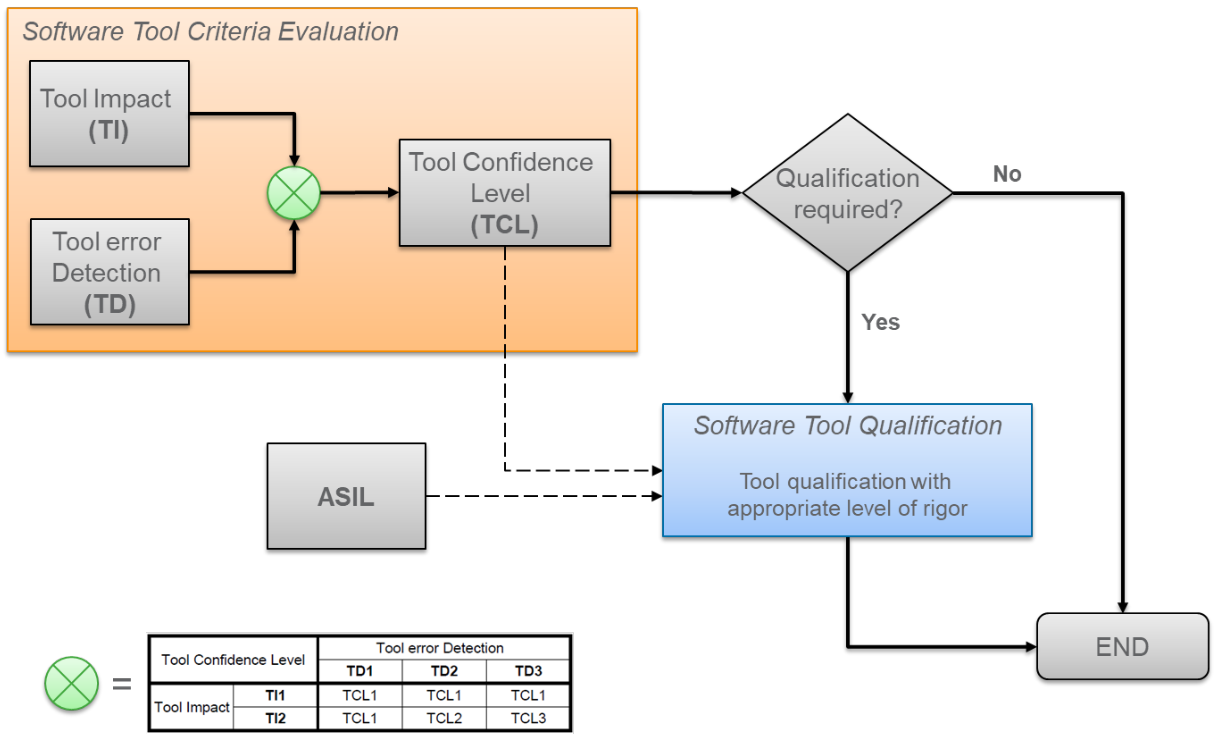
According to ISO 26262, to determine the required level of confidence in a software tool that is used in the development of a safety-related automotive product, the following criteria are evaluated:

- The possibility that the malfunctioning software tool and its corresponding erroneous output can introduce or fail to detect errors in a safety-related element being developed.
- The confidence in preventing or detecting such errors in its corresponding output.

This procedure is called Software Tool Criteria Evaluation, and it must be performed for all software tools that are involved in the development a safety-related element, resulting in a required Tool Confidence Level (TCL) for each software tool.

If the software tool criteria evaluation determines that a medium or high TCL is required, then appropriate Software Qualification methods must be applied, effectively reducing the risk of a critical software tool error. The choice of software qualification methods depends on the required TCL and the maximum ASIL of all the safety requirements allocated to the element developed using the software tool. However, if the software tool criteria evaluation determines that only a low TCL is required, then there is no need to apply such software qualification methods.

The software tool criteria evaluation and software tool qualification flow are summarized in Figure 1.


Software Tool Criteria Evaluation
Tool Impact (TI)
Tool error Detection (TD)
Tool Confidence Level (TCL)
Qualification required?
No
Yes
Software Tool Qualification
Tool qualification with appropriate level of rigor
ASIL
END
=

Tool Confidence Level		Tool error Detection		
		TD1	TD2	TD3
Tool Impact	TI1	TCL1	TCL1	TCL1
	TI2	TCL1	TCL2	TCL3

Figure 1: Software tool criteria evaluation and software tool qualification flow

Work Split Between Synopsys and Tool Users

A software tool criteria evaluation must always be performed in the development environment of the final tool user, and in the context of the actual product development. It is in this context, where potential tool malfunctions, their effect on the safety-related product, and the effectiveness of prevention and detection measures must be analyzed.

However, the tool vendor can support the tool user by performing a software tool criteria evaluation (and, if required, a software tool qualification) on their own, based on assumed tool use cases and an assumed development environment. If the assumptions made by the tool vendor match the actual situation at the tool user, then the user can take over the evaluation (and qualification) results from the tool vendor. Besides significantly reducing the effort for the tool user, this approach can also result in a better quality for the software tool criteria evaluation and qualification, since the tool vendor typically has a more detailed understanding of the inner working and possible malfunctions of the software tool.

Synopsys has adopted exactly this approach, which is summarized in Figure 2.

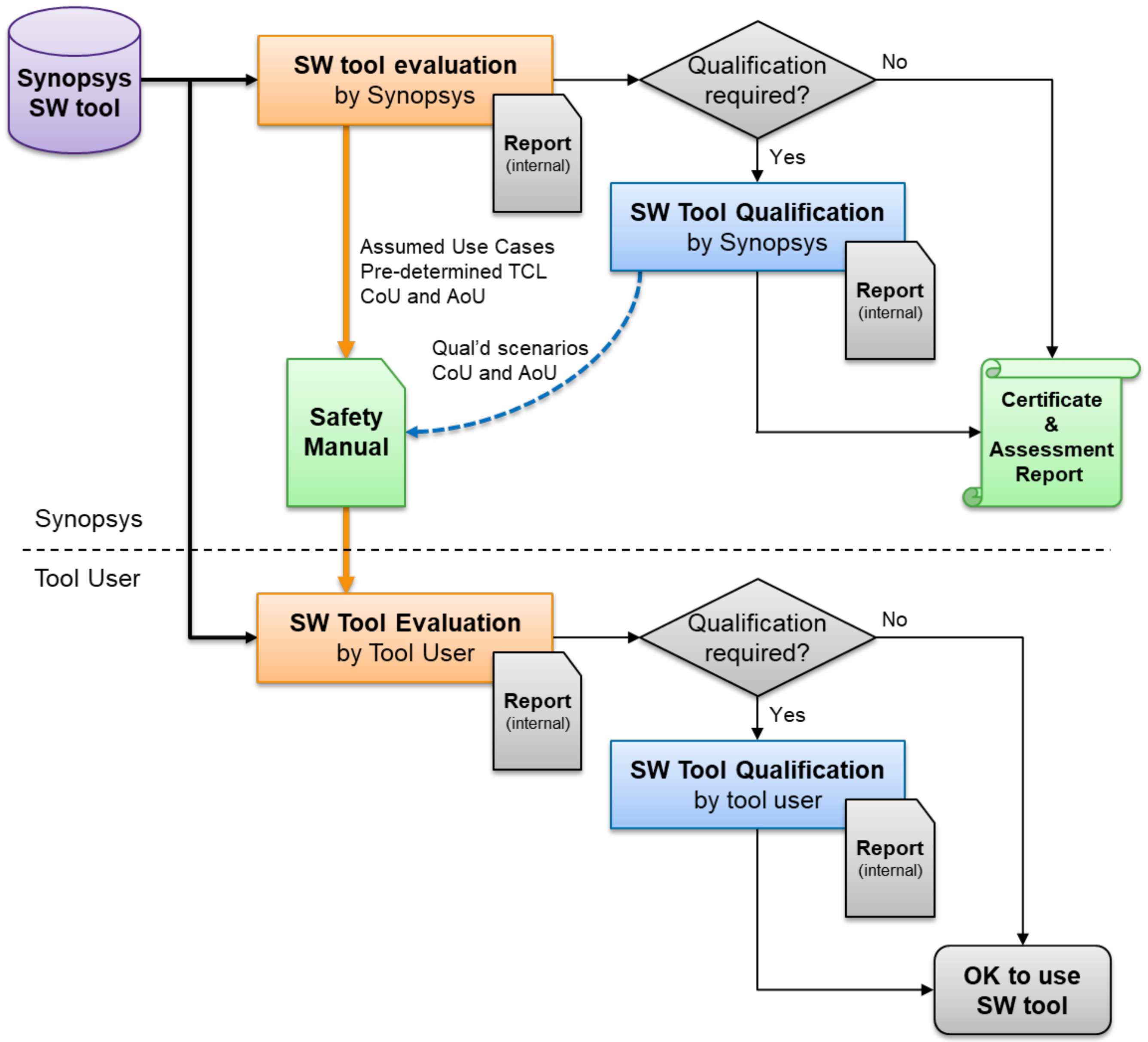


Figure 2: Work Split Between Synopsys and Tool Users

Synopsys performs the following activities:

1. Software tool criteria evaluation
    - Identification of possible **use cases** for the software tool, together with required **inputs** and expected **outputs**
    - Specification of **conditions of use (CoU)** for each use case, related to the development environment in which the tool is assumed to be deployed, including tool usage procedures and constraints
    - Analysis of potential software tool **malfunctions**, and their effect on a safety-related product that is developed with this tool
    - Analysis of **prevention** and **detection measures** internal to the software tool, to avoid tool malfunctions, or to control and mitigate their effects

- Specification of **assumptions of use (AoU)**, which are additional prevention and detection measures assumed to be performed by the end user of the tool
- Estimation of the **Tool Impact (TI)** for each malfunction, and the probability of **Tool error Detection (TD)** by the prevention and detection mechanisms (including assumptions of use)
- Determination of the required **Tool Confidence Level (TCL)** for each software tool malfunction, based on TI and TD
- Determination of the maximum TCL from all software tool malfunctions related to a use case. This is called the **pre-determined TCL** for the software tool use case
- Summary of the results in a software tool criteria evaluation report

2. Software tool qualification
   - If the pre-determined TCL indicates, that a medium (TCL2) or high (TCL3) tool confidence level is required for the software tool, then Synopsys may decide to perform a software tool qualification
   - The specific methods applied for tool qualification can vary for different tools and use cases, and they may include an evaluation of the software tool development process, the validation of the complete software tool, the validation of critical tool malfunctions with insufficient prevention and detection measures, or other methods
   - Summary of the qualification methods, procedures and results in a software tool qualification report

3. Safety manual for the software tool
   - The *Design Compiler Functional Safety Manual* (this document) is an important deliverable to the tool users, as it includes all end user-relevant information from the Synopsys software tool criteria evaluation and qualification
   - Software tool criteria evaluation related information, documented in Section 6, includes:
     - Description of software tool use cases
     - Description of the required inputs and expected outputs for each use case
     - Specification of conditions of use (CoU – conditions of the design, software tool, design environment, or situation that are assumed and required to be fulfilled by the user) for each use case
     - Specification of assumptions of use (AoU – actions that are assumed and required to be taken by the user of a software tool) for each use case
     - Pre-determined TCL for each use case
   - Software tool qualification related information (not required for this Design Compiler and therefore not included in this safety manual)
     - Description of the scope of the software tool qualification, including malfunctions and scenarios covered by the qualification
     - Specification of additional conditions of use (CoU) derived from the software tool qualification
     - Specification of additional assumptions of use (AoU) derived from the software tool qualification
   - Other information included in this safety manual
     - General information about the software tool needed by the tool user (see Appendix A)
     - Known limitations of the software tool, related to the described use cases as documented in Section 7

4. Certification and assessment report
    - Synopsys may decide to perform a functional safety assessment, to confirm the correctness, completeness and ISO 26262 conformance of the performed software tool criteria evaluation and qualification
    - Synopsys may also decide to achieve certification from an accredited third-party certification body, in addition to the functional safety assessment
    - The results of these activities are summarized in a functional safety assessment report and a certificate which can be viewed at exida Certificate for ISO 26262 Compliance

If the tool user wants to benefit from the work done by Synopsys, then according to the Figure 2 above, the user shall perform the following activities for each software tool:

1. Software tool criteria evaluation
    - Review and verify that the software tool criteria evaluation (and qualification) performed by Synopsys, as documented in the tool's Functional Safety Manual, matches the actual situation of the user's product development process
        - Verify whether the actual use case(s) of the software tool match those evaluated by Synopsys
        - Verify whether the actual inputs and outputs are identical to or a sub-set of those as evaluated by Synopsys
        - Verify that all conditions of use (CoU) specified by Synopsys are met, or whether the development process can be adjusted to meet these CoU(s)
        - Verify that all assumptions of use (AoU) specified by Synopsys are met, or whether the development process can be adjusted to meet these AoU(s)
        - Verify that the pre-determined Tool Confidence Level (TCL) for the relevant use case(s) are TCL1, *or*
        - Verify that Synopsys has successfully performed an additional software tool qualification for all TCL2 and TCL3 scenarios to conclude that the tool is suitable to be used for the development of a safety-related element of the same or higher ASIL than required by the user
    - If all the verification steps described above are successful, then the results of the Synopsys software tool criterial evaluation (and qualification) are applicable to the tool user, which means:
        - The required TCL pre-determined by Synopsys can be taken over by the tool user for actual product development
        - If the pre-determined TCL is TCL1, then the tool can be used without the need to perform any additional software tool qualification
        - If the pre-determined TCL is TCL2 or TCL3, then the software tool qualification performed by Synopsys is sufficient, and the tool can be used without the need for further software tool qualification by the end user
    - All of the steps above must be documented in a software tool criteria evaluation report, including evidence for the successful conclusion of all verification steps, which may include reference to the Synopsys Functional Safety Manual, and optionally, to the Synopsys certification and assessment report

2. Software tool qualification
    - If any of the verification steps described above as part of the tool user’s software tool criteria evaluation fails (e.g. different use case, CoU or AoU cannot be met, pre-determined TCL is not TCL1 and Synopsys has not performed a software tool qualification), then the user must perform his/her own software tool qualification
    - The specific methods applied for tool qualification are decided and planned by the tool user -- Synopsys does not recommend any specific methods or procedures
    - The summary of the qualification methods, procedures and results shall be documented in a software tool qualification report

4 Design Compiler Description

This section provides a general description regarding the use of the Design Compiler tool as a software tool in the development of safety-related applications and describes where to get the latest product documentation and the runtime environment required to use the Design Compiler tool.

Coverage

The *Design Compiler Functional Safety Manual* is intended to be used starting with the version N-2017.09 and later versions of the Design Compiler tool per the use cases presented in this document. In general, unless otherwise noted, the failure modes and detection mechanisms noted in the use cases presented in Section 6 are tool version independent.

Compliance with ISO 26262

The Design Compiler tool can be used in the development of safety-related elements according to ISO 26262, with allocated safety requirements up to a maximum Automotive Safety Integrity Level D (ASIL D), if the tool is used in the context of a tool chain and in compliance with this document, the *Design Compiler Functional Safety Manual*.

See the exida Certificate for ISO 26262 Compliance of Synopsys Design Compiler when used in a tool chain flow.

Product Documentation and Support

Comprehensive documentation for using the Design Compiler tool is provided on SolvNetPlus. The latest documentation for the Design Compiler tool can be accessed at Design Compiler Online Help on SolvNetPlus.

Specific documentation for performing design and analysis as part of an ISO 26262 compliant flow is provided in Section 3, Section 5, Section 6 and Appendix A of this document, the *Design Compiler Functional Safety Manual*.

Synopsys provides online customer support for the Design Compiler tool. See Section 1 for more information.

Installation and Supported Platforms

The installation of the Design Compiler tool must follow the guidelines in the *Synopsys® Installation Guide* as well as the specific *Design Compiler Installation Notes* document.

Users are required to download the tool executable and INSTALL_README from the SolvNetPlus site at https://solvnet.synopsys.com/DownloadCenter/dc/product.jsp.

Supported platforms and operating systems requirements:

- For installation instructions, see the *Synopsys® Installation Guide* at https://www.synopsys.com/install.
- For the latest supported binary-compatible hardware platform or operating system, including required operating system patches, see https://www.synopsys.com/qsc.
- If updates (including security patches) to computing environments (including operating systems) are backward compatible with previous versions of the computing environment used to test the Design Compiler tool, the results of the testing performed by Synopsys using such previous versions are applicable.

Additional information:

- For information about the compute platforms roadmap, go to https://www.synopsys.com/support/licensing-installation-computeplatforms/compute-platforms/compute-platforms-roadmap.html.
- For platform notices, go to https://www.synopsys.com/support/licensing-installation-computeplatforms/compute-platforms/platform-notice.html.
- For information regarding the license key retrieval process, go to https://solvnet.synopsys.com/smartkeys/smartkeys.cgi.

User Competence

To properly use the Design Compiler tool, a user must have a good understanding and working knowledge of the following:

- Electrical engineering and circuit design
- The ISO 26262 standard
- Documentation of the Design Compiler tool, such as the User Guide, at Design Compiler Online Help on SolvNetPlus.
- This Functional Safety Manual
- The published list of safety-related defects for the Design Compiler tool available at Design Compiler Safety-Related Issues Master List.
- Applicability of the Design Compiler tool in the overall tool chain

Managing Known Safety-Related Defects

Synopsys maintains current information for every reported defect through STARs. The Design Compiler team evaluates each reported issue for potential impact on functional safety.

A list of all known safety-related defects for each release of Design Compiler is available on a SolvNetPlus knowledge base article and is referenced from the *Design Compiler Release Notes*.

Design Compiler users must assess, as part of their own software tool criteria evaluation, the potential impact of the known safety-related defects in their design and must ensure mitigation of any relevant safety-related defects.

Managing New Releases

Synopsys can release new versions of the Design Compiler tool at any time to extend its functionality or to fix defects. When a new version is available, notification is posted on the SolvNetPlus site. A subscription service is available for users to be notified of any new product releases.

When installing a new version of the Design Compiler tool, users must evaluate the impact of any known safety-related defects in their design by checking the accompanying *Design Compiler Release Notes* for the following:

- Any changes that apply to safety-related use cases
- List of known safety-related defects in the new version of the Design Compiler tool

In addition, users must refer to the latest version of this document, the *Design Compiler Functional Safety Manual*, available with the product release contents.

5 Synopsys Digital Tool Chain

This section provides an overview of where the Design Compiler is used in the tool chain.

The ISO 26262 standard provides a methodology and requirements for software tool criteria evaluation and qualification (see ISO 26262-8, clause 11). It applies to software tools used for the development of safety-related designs where it is essential that the tool operates correctly without introducing or failing to detect errors in the safety-related design.

The suitability of a software tool to be used in the development of a safety-related design is determined in the software tool criteria evaluation, which results in a Tool Confidence Level (TCL): a level of confidence that the software tool does not introduce or fail to detect an error in the design without being noticed, and mitigated before the design is released as a safety-related product. This evaluation is best performed in the context of the overall software tool chain and development flow, in which the individual software tool is used. The following high-level diagram reflects the tool chain for which the Design Compiler tool is applicable.

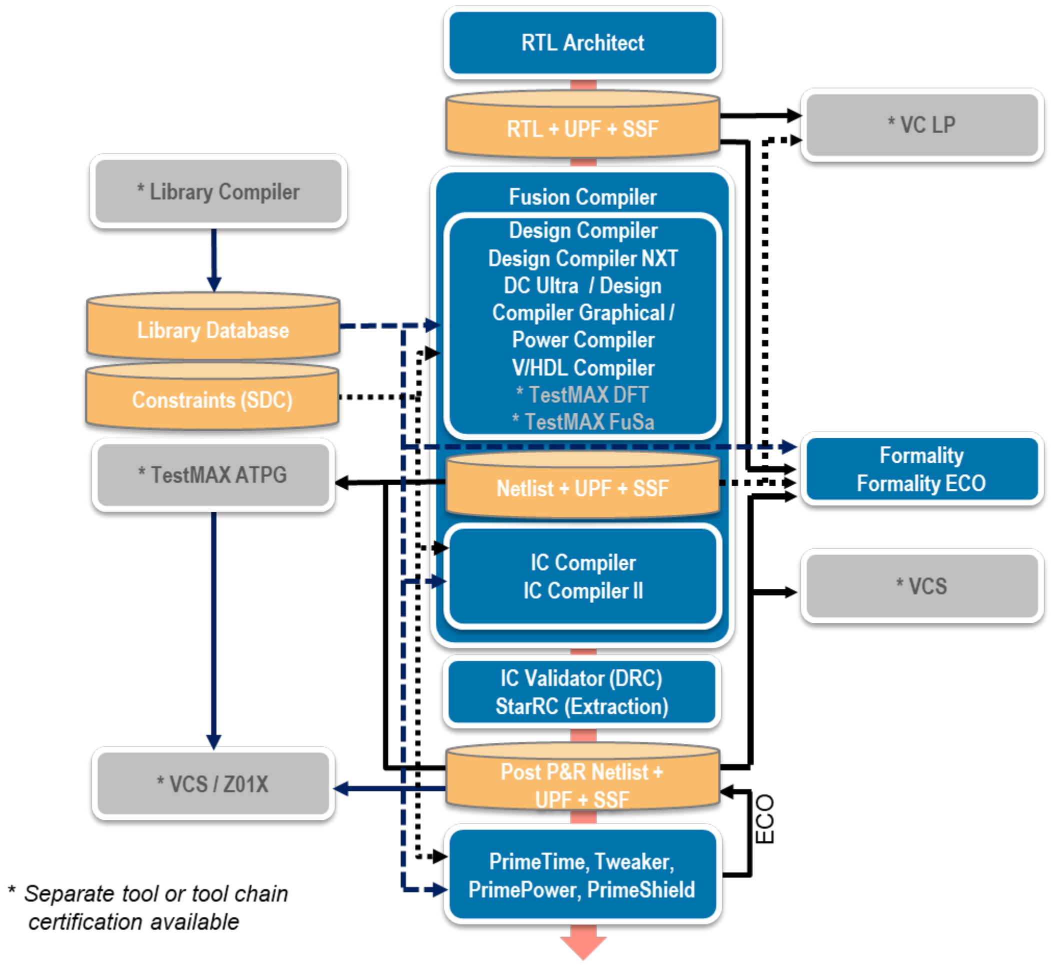


Figure 3: Synopsys Digital Tool Chain

6
Use Cases

This section describes the safety-qualified use cases of the Design Compiler tool. Users should also perform TCL determination based on their specific Use Cases.

The Design Compiler tool enables the user to take a functional description of their design written in RTL (SystemVerilog or VHDL), translate it to a logically equivalent gate level description of the design and optimize it for a variety of user provided constraints. This section describes use cases for several different sections of the Design Compiler tool:

- HDL Compiler
  - This is the initial step of translating a RTL description of the design functionality into a gate level representation. This step should be executed before wireload-based technology mapping and logic optimization or physical-based mapping and optimization.
  - Affected Tools
    - HDL Compiler
- Wireload-Based Technology Mapping and Logic Optimization
  - This is the step of translating the GTECH output of the HDL Compiler step into a user provided technology library and optimizing the design based on user constraints. This step is done when no physical based information is available.
  - Affected Tools
    - Design Compiler
    - Design Compiler NXT
    - Design Compiler Graphical
    - DC-Ultra
    - Power Compiler
- Physical-Based Mapping and Optimization
  - This step can be done instead of wireload-based technology mapping and logic optimization when physical information (library and floorplan) are available. The functionality and failure modes are a superset of wireload-based technology mapping and logic optimization.
  - Affected Tools
    - Design Compiler NXT
    - Design Compiler Graphical
    - Power Compiler

- Physical-Based Mapping and Optimization with a link-to-IC Compiler II
    - This step can be done instead of physical-based mapping and optimization when greater correlation with the IC Compiler II tool is required or support for small geometries is required using the Design Compiler NXT tool. The functionality and failure modes are a superset of wireload-based technology mapping and logic optimization and physical-based mapping and optimization.
    - Affected Tools
        - Design Compiler NXT
- Multi-Voltage Synthesis
    - This is an additional design and optimization step that can be run on top of wireload-based technology mapping and logic optimization or physical-based mapping and optimization, covering designs with shut-down regions and/or power/ground supplies operating at different or multiple voltages. Assumptions of use for wireload or physical synthesis apply in addition to the assumptions unique to this mode.
    - Affected Tools
        - Design Compiler NXT
        - Design Compiler Graphical
        - Power Compiler
- Functional Safety Register Synthesis
    - This is an additional design and optimization step that can be run on top of wireload-based technology mapping and logic optimization or physical-based mapping and optimization, covering designs with safety mechanisms in the form of safety registers such as TMR, DMR, and fault-tolerant registers. Assumptions of use for wireload or physical synthesis apply in addition to the assumptions unique to this mode.
    - Affected Tools
        - Design Compiler NXT
- Functional Safety Core Synthesis
    - This is an additional design and optimization step that can be run on top of wireload-based technology mapping and logic optimization or physical-based mapping and optimization, covering designs with safety mechanisms in the form of safety cores, such as DCLS. Assumptions of use for wireload or physical synthesis apply in addition to the assumptions unique to this mode.
    - Affected Tools
        - Design Compiler NXT

- Failsafe Finite State Machine (FSM) Synthesis
    - This is an additional design and optimization step that can be run on top of wireload-based technology mapping and logic optimization or physical-based mapping and optimization, covering designs with safety mechanisms in the form of failsafe finite state machines. Assumptions of use for wireload or physical synthesis apply in addition to the assumptions unique to this mode.
    - Affected Tools
        - Design Compiler NXT
- Power Optimization and Clock Gating
    - This is an additional design and optimization step that can be run on top of wireload-based technology mapping and logic optimization or physical-based mapping and optimization, covering designs with constraints optimizing for power consumption (dynamic and static) or commands to implement clock gating. Assumptions of use for wireload or physical synthesis apply in addition to the assumptions unique to this mode.
    - Affected Tools
        - Design Compiler
        - Design Compiler NXT
        - Design Compiler Graphical
        - DC-Ultra
        - Power Compiler
- Gate-to-Gate Optimization
    - This is a stand-alone optimization mode intended for designs that have already gone through the above use cases (either in Design Compiler or a 3rd party synthesis tool). The assumptions of use for this mode are independent of the other modes.

Use Case 1: HDL Compiler

In this use case, the user's main goal is to perform translation of RTL to GTECH (Design Compiler's internal design representation). This is the initial step in the synthesis process. All designs should execute this use case.

In this use case, the HDL Compiler tool uses and generates the following main inputs and outputs.

- Affected Tools
    - HDL Compiler

- Inputs:
    - RTL (.v, .sv, .vhd) text file
    - TCL script (configuration, run script)
- Expected outputs:
    - GTECH netlist (.ddc)
    - Elaboration report (log)
- Related commands: analyze, elaborate, read_verilog, read_vhdl, read_file

For this use case of *Design Compiler*, the following conditions of use (constraints for the design and design environment, recommended procedures for the tool usage, etc.) shall be met:

- CoU- DC-001: User shall review all error and warning messages and take appropriate action.
- CoU- DC-002: User shall follow the Design Compiler Reference Methodology or use equivalent scripts.
- CoU-DC-003: For the final run, Tcl script-based batch mode execution shall be used, without interactive command line entry or GUI manual command entry. Tcl scripts and log files shall be retained as design signoff records.

For this use case of *Design Compiler*, the following assumptions of use (required actions to be taken by the tool user to prevent or detect design errors due to possible tool malfunctions) shall be met:

- AoU-DC-001: User shall review synthesis log for errors, unresolved references, unmapped cells and expected results (log must include elaboration report).
- AoU-DC-002: User shall check that all outputs are generated with an up-to-date timestamp.
- AoU-DC-003: User shall verify post-synthesis netlist vs. RTL with a formal equivalence tool (such as Formality). The netlist verified should be the same netlist and format that is to be used in later implementation and verification steps.
- AoU-DC-004: User shall review synthesis log for errors, unresolved references, unmapped cells and expected results (log must include output from the following commands: check_design, report_timing, report_area, compile/compile_ultra, and report_reference).
- AoU-DC-005: User shall review paths with infer_mux in GTECH netlist to ensure MUX_OP is used for the selector.
- AoU-DC-006: User shall verify synthesis or place and route netlist by gate-level simulation for correct reset behavior using a logic simulator (such as VCS).

All analyzed failure modes and prevention, detection and mitigation measures (including conditions and assumptions of use listed above) are independent of the exact Design Compiler tool version.

A software tool criteria evaluation performed by Synopsys according to ISO 26262-8, clause 11, which assumes the fulfillment of all conditions of use (CoU) and assumptions of use (AoU) as described above, results in a required tool confidence level:

TCL1 for *Design Compiler* Use Case 1: HDL Compiler

In this case, no further activities for software tool qualification are required.

Use Case 2: Wireload-Based Technology Mapping and Logic Optimization

In this use case, the user's main goal is to map the generic technology netlist (GTECH) to a technology specific representation of the design and optimize for provided constraints like timing and area. In this use case, no physical data (such as a floorplan) is provided to the tool.

In this use case, the Design Compiler tool uses and generates the following main inputs and outputs.

- Affected Tools
  - Design Compiler
  - Design Compiler NXT
  - Design Compiler Graphical
  - DC-Ultra
  - Power Compiler
- Inputs:
  - GTECH netlist (.ddc) (This is the output from the HDL Compiler step)
  - Technology library (.db) (Provided by ASIC vendor)
  - User constraints (SDC) (User generated)
  - Tcl script (configuration, run script) (User generated)
- Expected outputs:
  - Mapped Netlist (.ddc)
  - log file (.txt)
  - analyzed tech library (alib)
    - This is an internal file generated by Design Compiler. It contains an analyzed version of the technology library input
- Related commands: compile, compile_ultra

For this use case of *Design Compiler*, the following conditions of use (constraints for the design and design environment, recommended procedures for the tool usage, etc.) shall be met:

- CoU- DC-001: User shall review all error and warning messages and take appropriate action.
- CoU- DC-002: User shall follow the Design Compiler Reference Methodology or use equivalent scripts.
- CoU-DC-003: For the final run, Tcl script-based batch mode execution shall be used, without interactive command line entry or GUI manual command entry. Tcl scripts and log files shall be retained as design signoff records.
- CoU-DC-004: Pre-route timing and power calculations are not considered accurate enough for final design signoff. User shall validate timing and power requirements post-route in an appropriate signoff tools (such as PrimeTime and PrimePower) prior to delivery of the final product.

For this use case of *Design Compiler*, the following assumptions of use (required actions to be taken by the tool user to prevent or detect design errors due to possible tool malfunctions) shall be met:

- AoU-DC-002: User shall check that all outputs are generated with an up-to-date timestamp.
- AoU-DC-003: User shall verify post-synthesis netlist vs. RTL with a formal equivalence tool (such as Formality). The netlist verified should be the same netlist and format that is to be used in later implementation and verification steps.
- AoU-DC-004: User shall review synthesis log for errors, unresolved references, unmapped cells and expected results (log must include output from the following commands: check_design, report_timing, report_area, compile/compile_ultra, and report_reference).
- AoU-DC-006: User shall verify synthesis or place & route netlist by gate-level simulation for correct reset behavior using a logic simulator (such as VCS).
- AoU-DC-007: User shall verify synthesis netlist timing in a signoff static timing tool (such as PrimeTime).
- AoU-DC-008: User shall review logs of DFT tool (such as DFT Compiler or TestMAX DFT) for errors and verify scan coverage.
- AoU-DC-010: User shall review paths with infer_mux in synthesis netlist to ensure mux cells are used.
- AoU-DC-012: User shall review place and route tool (such as IC Compiler or IC Compiler II) log file for errors during read (place and route is the next implementation step in the digital tool chain).
- AoU-DC-013: User shall review log of clock domain crossing checking tools (such as SpyGlass CDC).

All analyzed failure modes and prevention, detection and mitigation measures (including conditions and assumptions of use listed above) are independent of the exact Design Compiler tool version.

A software tool criteria evaluation performed by Synopsys according to ISO 26262-8, clause 11, which assumes the fulfillment of all conditions of use (CoU) and assumptions of use (AoU) as described above, results in a required tool confidence level:

TCL1 for *Design Compiler* Use Case 2: Wireload-Based Technology Mapping and Logic Optimization

In this case, no further activities for software tool qualification are required.

Use Case 3: Physical-Based Mapping and Optimization

In this use case, the user's main goal is to map the generic technology netlist (GTECH) to a technology specific representation of the design and optimize for provided constraints like timing and area. In this use case, physical data (such as a floorplan) is provided to the tool and optimization takes place in the context of standard cell placement.

Note: These settings are applied on top of use cases 2, only inputs and outputs unique to this use case are listed.

In this use case, the Design Compiler tool uses and generates the following main inputs and outputs.

- Affected Tools
    - Design Compiler NXT
    - Design Compiler Graphical
    - Power Compiler
- Inputs:
    - Physical libraries (Milkyway)
    - Floorplan information (DEF)
    - Inputs of Use Case 2
- Expected outputs:
    - If physical information is passed to the place and route tool, it will be included with the design database (.ddc output)
    - Outputs from Use Case 2
- Related commands: compile, compile_ultra

For this use case of *Design Compiler*, the following conditions of use (constraints for the design and design environment, recommended procedures for the tool usage, etc.) shall be met:

- CoU- DC-001: User shall review all error and warning messages and take appropriate action.
- CoU- DC-002: User shall follow the Design Compiler Reference Methodology or use equivalent scripts.
- CoU-DC-003: For the final run, Tcl script-based batch mode execution shall be used, without interactive command line entry or GUI manual command entry. Tcl scripts and log files shall be retained as design signoff records.
- CoU-DC-004: Pre-route timing and power calculations are not considered accurate enough for final design signoff. User shall validate timing and power requirements post-route in an appropriate signoff tools (such as PrimeTime and PrimePower) prior to delivery of the final product.

For this use case of *Design Compiler*, the following assumptions of use (required actions to be taken by the tool user to prevent or detect design errors due to possible tool malfunctions) shall be met:

- AoU-DC-002: User shall check that all outputs are generated with an up-to-date timestamp.
- AoU-DC-003: User shall verify post-synthesis netlist vs. RTL with a formal equivalence tool (such as Formality). The netlist verified should be the same netlist and format that is to be used in later implementation and verification steps.
- AoU-DC-004: User shall review synthesis log for errors, unresolved references, unmapped cells and expected results (log must include output from the following commands: check_design, report_timing, report_area, compile/compile_ultra, and report_reference).
- AoU-DC-006: User shall verify synthesis or place & route netlist by gate-level simulation for correct reset behavior using a logic simulator (such as VCS).
- AoU-DC-007: User shall verify synthesis netlist timing in a signoff static timing tool (such as PrimeTime).
- AoU-DC-008: User shall review logs of DFT tool (such as DFT Compiler or TestMAX DFT) for errors and verify scan coverage.
- AoU-DC-010: User shall review paths with infer_mux in synthesis netlist to ensure mux cells are used.
- AoU-DC-011: User shall review floorplan visually in synthesis or place and route tool (such as Design Compiler NXT or IC Compiler or IC Compiler II).
- AoU-DC-012: User shall review place and route tool (such as IC Compiler or IC Compiler II) log file for errors during read (place and route is the next implementation step in the digital tool chain).
- AoU-DC-013: User shall review log of clock domain crossing checking tools (such as SpyGlass CDC).

All analyzed failure modes and prevention, detection and mitigation measures (including conditions and assumptions of use listed above) are independent of the exact Design Compiler tool version.

A software tool criteria evaluation performed by Synopsys according to ISO 26262-8, clause 11, which assumes the fulfillment of all conditions of use (CoU) and assumptions of use (AoU) as described above, results in a required tool confidence level:

TCL1 for *Design Compiler* Use Case 3: Physical Based Mapping and Optimization

In this case, no further activities for software tool qualification are required.

Use Case 4: Physical-Based Mapping and Optimization With Link to IC Compiler II

In this use case, the user's main goal is to map the generic technology netlist (GTECH) to a technology specific representation of the design and optimize for provided constraints like timing and area. In this use case, physical data (such as a floorplan) is provided to the tool and optimization takes place in the context of standard cell placement.

Note: This use case is a super-set of Use Case 2 and 3. Only inputs and outputs unique to this use case are listed.

In this use case, the Design Compiler tool uses and generates the following main inputs and outputs.

- Affected Tools
    - Design Compiler NXT
- Inputs:
    - IC Compiler II compatible physical library information (NDM)
    - IC Compiler II setup and configuration script (Tcl)
    - Inputs of Use Case 2 or 3
- Expected outputs:
    - If physical information is passed from the place and route tool back to the Design Compiler NXT tool, it will be included with the design database (.ddc output)
    - Outputs from Use Case 2 or 3
- Related commands: compile, compile_ultra

For this use case of *Design Compiler NXT*, the following conditions of use (constraints for the design and design environment, recommended procedures for the tool usage, etc.) shall be met:

- CoU- DC-001: User shall review all error and warning messages and take appropriate action.
- CoU-DC-003: For the final run, Tcl script-based batch mode execution shall be used, without interactive command line entry or GUI manual command entry. Tcl scripts and log files shall be retained as design signoff records.
- CoU-DC-004: Pre-route timing and power calculations are not considered accurate enough for final design signoff. User shall validate timing and power requirements post-route in an appropriate signoff tools (such as PrimeTime and PrimePower) prior to delivery of the final product.
- CoU-DC-006: User shall verify synthesis or place and route netlist by gate-level simulation for correct reset behavior using a logic simulator (such as VCS).

In this use case, the IC Compiler II tool is called from *Design Compiler NXT* for the purpose of improving correlation between the tools. The functionality invoked in the IC Compiler II tool is a small subset of the complete functionality of the IC Compiler II tool, and as such, only the following Conditions of Use from the IC Compiler II tool apply to this Use Case and are listed for completeness:

- CoU-ICCII-001: User shall review all error and warning messages and take appropriate action.

For this use case of *Design Compiler NXT*, the following assumptions of use (required actions to be taken by the tool user to prevent or detect design errors due to possible tool malfunctions) shall be met. The assumptions of use for Use Cases 2 and 3 apply to this use case as well and are required. The functionality executed in the IC Compiler II tool is reflected in the *Design Compiler NXT* log file. Therefore, any issues in the functionality of the IC Compiler II tool can be detected in the log file:

- AoU-DC-002: User shall check that all outputs are generated with an up-to-date timestamp.
- AoU-DC-004: User shall review synthesis log for errors, unresolved references, unmapped cells and expected results (log must include output from the following commands: check_design, report_timing, report_area, compile/compile_ultra, and report_reference).
- AoU-DC-007: User shall verify synthesis netlist timing in a signoff static timing tool (such as PrimeTime).

As mentioned above, the functionality invoked in the IC Compiler II tool is a small subset of the complete functionality of the IC Compiler II tool, and as such, only the following Assumptions of Use from the IC Compiler II tool apply to this Use Case. The required actions are covered by the existing Design Compiler Assumptions of Use, but the following IC Compiler II AoUs are listed for completeness:

- AoU-ICCII-001: User shall review the log files for error messages, warning messages, correct, complete, or unexpected flow execution, and random characters.
- AoU-ICCII-002: User shall review all relevant QoR reports, such as timing and power reports, for error messages, warning messages, random characters, and expected results.
- AoU-ICCII-004: User shall perform independent formal equivalence checking of the gate-level netlist before and after place and route using a formal verification tool (such as Formality).
- AoU-ICCII-022: User shall review the log files for error messages, warning messages, and correct operation during the saving and subsequent reloading of the NDM libraries.

All analyzed failure modes and prevention, detection and mitigation measures (including conditions and assumptions of use listed above) are independent of the exact Design Compiler tool version.

A software tool criteria evaluation performed by Synopsys according to ISO 26262-8, clause 11, which assumes the fulfillment of all conditions of use (CoU) and assumptions of use (AoU) as described above, results in a required tool confidence level:

TCL1 for *Design Compiler* Use Case 4: Physical Based Mapping and Optimization With Link to IC Compiler II

In this case, no further activities for software tool qualification are required.

Use Case 5: Multi-Voltage Synthesis

In this use case, the user's main goal is to add constraints and design information based on multi-voltage design styles such as multiple voltages / supply nets, shutdown areas, or voltage scaling.

Note: These settings are applied on top of use cases 2 or 3, the detection measures conditions and assumptions of use (CoU and AoU) for this use case are applied in addition to the detection measures for those use cases.

In this use case, the Design Compiler tool uses and generates the following main inputs and outputs.

- Affected Tools
  - Design Compiler NXT
  - Design Compiler Graphical
  - Power Compiler
- Inputs:
  - Power intent (UPF)
  - Inputs of Use Case 2 or 3

- Expected outputs:
  - Incremental or updated UPF (UPF')
  - Outputs from Use Case 2 or 3
- Related commands: load_upf prior to compile/compile_ultra

For this use case of *Design Compiler*, the following conditions of use (constraints for the design and design environment, recommended procedures for the tool usage, etc.) shall be met:

- CoU- DC-001: User shall review all error and warning messages and take appropriate action.
- CoU- DC-002: User shall follow the Design Compiler Reference Methodology or use equivalent scripts.
- CoU-DC-003: For the final run, Tcl script-based batch mode execution shall be used, without interactive command line entry or GUI manual command entry. Tcl scripts and log files shall be retained as design signoff records.
- CoU-DC-004: Pre-route timing and power calculations are not considered accurate enough for final design signoff. User shall validate timing and power requirements post-route in an appropriate signoff tools (such as PrimeTime and PrimePower) prior to delivery of the final product.
- CoU-DC-005: User shall run load_upf prior to issuing the compile/compile_ultra command.

For this use case of *Design Compiler*, the following assumptions of use (required actions to be taken by the tool user to prevent or detect design errors due to possible tool malfunctions) shall be met:

- AoU-DC-002: User shall check that all outputs are generated with an up-to-date timestamp.
- AoU-DC-009: User shall review synthesis log for errors, unresolved references, unmapped cells and expected results (log must include output from the following commands: check_design, report_timing, report_area, compile/compile_ultra, and check_mv_design).
- AoU-DC-012: User shall review place and route tool (such as IC Compiler or IC Compiler II) log file for errors during read (place and route is the next implementation step in the digital tool chain).
- AoU-DC-014: User shall review low power static checking tool results (such as VC LP) run on gate-level netlist plus UPF'. This will compare original intent vs UPF' output intent.
- AoU-DC-018: User shall verify post-synthesis netlist vs. RTL with a formal equivalence tool (such as Formality) including UPF verification. The netlist verified should be same netlist and format that is to be used in later implementation and verification steps.
- AoU-DC-019: User shall verify synthesis netlist timing in a signoff static timing tool (such as PrimeTime) including voltage violations.

All analyzed failure modes and prevention, detection and mitigation measures (including conditions and assumptions of use listed above) are independent of the exact Design Compiler tool version.

A software tool criteria evaluation performed by Synopsys according to ISO 26262-8, clause 11, which assumes the fulfillment of all conditions of use (CoU) and assumptions of use (AoU) as described above, results in a required tool confidence level:

TCL1 for *Design Compiler* Use Case 5: Multi-Voltage Synthesis

In this case, no further activities for software tool qualification are required.

Use Case 6: Functional Safety Register Synthesis

In this use case, the user's main goal is to add safety mechanisms in the form of safety registers, such as triple-modular redundancy (TMR), dual-modular redundancy (DMR), and fault tolerant registers, to mitigate the impact of single event upsets (SEU) due to transient faults such as those coming from ion or electromagnetic radiation.

Note: These settings are applied on top of use cases 2 or 3, the detection measures conditions and assumptions of use (CoU and AoU) for this use case are applied in addition to the detection measures for those use cases.

In this use case, the Design Compiler tool uses and generates the following main inputs and outputs.

- Affected Tools
  - Design Compiler NXT
- Inputs:
  - Safety Intent (Safety Specification Format (SSF) or Tcl)
  - Single point fault metric (SPFM) loss calculations on registers (Tcl)
  - Inputs of Use Case 2 or 3
- Expected outputs:
  - Modified or incremental safety intent (SSF')
  - Outputs of Use Case 2 or 3
- Related commands: load_ssf, save_ssf in use with compile/compile_ultra

For this use case of *Design Compiler*, the following conditions of use (constraints for the design and design environment, recommended procedures for the tool usage, etc.) shall be met:

- CoU- DC-001: User shall review all error and warning messages and take appropriate action.
- CoU- DC-002: User shall follow the Design Compiler Reference Methodology or use equivalent scripts.

- CoU-DC-003: For the final run, Tcl script-based batch mode execution shall be used, without interactive command line entry or GUI manual command entry. Tcl scripts and log files shall be retained as design signoff records.
- CoU-DC-004: Pre-route timing and power calculations are not considered accurate enough for final design signoff. User shall validate timing and power requirements post-route in an appropriate signoff tools (such as PrimeTime and PrimePower) prior to delivery of the final product.

For this use case of *Design Compiler*, the following assumptions of use (required actions to be taken by the tool user to prevent or detect design errors due to possible tool malfunctions) shall be met:

- AoU-DC-002: User shall check that all outputs are generated with an up-to-date timestamp.
- AoU-DC-009: User shall review synthesis log for errors, unresolved references, unmapped cells and expected results (log must include output from the following commands: check_design, report_timing, report_area, compile/compile_ultra, and check_mv_design).
- AoU-DC-022: When SSF has been utilized, user shall review the relevant safety reports (report_safety_status, report_safety_register_groups, report_fsm) and validate expected safety measures are implemented.
- AoU-DC-023: When SSF has been utilized, user shall validate correctness of any SSF output from the tool. (for example, generate safety reports; write SSF' and read it back into the tool; generate safety reports and compare).
- AoU-DC-024: When SSF has been utilized, user shall perform independent formal equivalence checking of RTL to post-synthesis netlist using a formal verification tool that supports safety intent checking (such as Formality) including SVF information and applicable safety intent. The netlist verified should be same netlist and format that is to be used in later implementation and verification steps.
- AoU-DC-025: User shall verify gate-level SPFM meets design ASIL requirement on post-synthesis netlist with a fault modeling tool (such as TestMAX FuSa or Z01X). The netlist verified should be same netlist and format that is to be used in later implementation and verification steps.

All analyzed failure modes and prevention, detection and mitigation measures (including conditions and assumptions of use listed above) are independent of the exact Design Compiler tool version.

A software tool criteria evaluation performed by Synopsys according to ISO 26262-8, clause 11, which assumes the fulfillment of all conditions of use (CoU) and assumptions of use (AoU) as described above, results in a required tool confidence level:

TCL1 for *Design Compiler* Use Case 6: Functional Safety Register Synthesis

In this case, no further activities for software tool qualification are required.

Use Case 7: Functional Safety Core Synthesis

In this use case, the user's main goal is to add safety mechanisms in the form of safety cores, such as Dual Core LockStep (DCLS), to mitigate the impact of SEUs due to transient faults such as those coming from ion or electromagnetic radiation.

Note: These settings are applied on top of use cases 2 or 3, the detection measures conditions and assumptions of use (CoU and AoU) for this use case are applied in addition to the detection measures for those use cases.

In this use case, the Design Compiler tool uses and generates the following main inputs and outputs.

- Affected Tools
  - Design Compiler NXT
- Inputs:
  - Safety Intent (SSF or Tcl)
  - Inputs of Use Case 2 or 3
- Expected outputs:
  - Modified or incremental safety intent (SSF')
  - Outputs of Use Case 2 or 3
- Related commands: load_ssf, save_ssf in use with compile/compile_ultra

For this use case of *Design Compiler*, the following conditions of use (constraints for the design and design environment, recommended procedures for the tool usage, etc.) shall be met:

- CoU- DC-001: User shall review all error and warning messages and take appropriate action.
- CoU- DC-002: User shall follow the Design Compiler Reference Methodology or use equivalent scripts.
- CoU-DC-003: For the final run, Tcl script-based batch mode execution shall be used, without interactive command line entry or GUI manual command entry. Tcl scripts and log files shall be retained as design signoff records.

- CoU-DC-004: Pre-route timing and power calculations are not considered accurate enough for final design signoff. User shall validate timing and power requirements post-route in an appropriate signoff tools (such as PrimeTime and PrimePower) prior to delivery of the final product.

For this use case of *Design Compiler*, the following assumptions of use (required actions to be taken by the tool user to prevent or detect design errors due to possible tool malfunctions) shall be met:

- AoU-DC-002: User shall check that all outputs are generated with an up-to-date timestamp.
- AoU-DC-009: User shall review synthesis log for errors, unresolved references, unmapped cells and expected results (log must include output from the following commands: check_design, report_timing, report_area, compile/compile_ultra, and check_mv_design).
- AoU-DC-011: User shall review floorplan visually in synthesis or place and route tool (such as Design Compiler NXT or IC Compiler or IC Compiler II).
- AoU-DC-022: When SSF has been utilized, user shall review the relevant safety reports (report_safety_status, report_safety_register_groups, report_fsm) and validate expected safety measures are implemented.
- AoU-DC-023: When SSF has been utilized, user shall validate correctness of any SSF output from the tool. (for example, generate safety reports; write SSF' and read it back into the tool; generate safety reports and compare).

All analyzed failure modes and prevention, detection and mitigation measures (including conditions and assumptions of use listed above) are independent of the exact Design Compiler tool version.

A software tool criteria evaluation performed by Synopsys according to ISO 26262-8, clause 11, which assumes the fulfillment of all conditions of use (CoU) and assumptions of use (AoU) as described above, results in a required tool confidence level:

TCL1 for *Design Compiler* Use Case 6: Functional Safety Core Synthesis

In this case, no further activities for software tool qualification are required.

Use Case 8: Failsafe Finite State Machine Synthesis

In this use case, the user's main goal is to add safety mechanisms in the form of failsafe finite state machines (FSM) to mitigate the impact of SEUs due to transient faults such as those coming from ion or electromagnetic radiation.

Note: These settings are applied on top of use cases 2 or 3, the detection measures conditions and assumptions of use (CoU and AoU) for this use case are applied in addition to the detection measures for those use cases.

In this use case, the Design Compiler tool uses and generates the following main inputs and outputs.

- Affected Tools
  - Design Compiler NXT
- Inputs:
  - RTL (.v, .sv, .vhd) text file
  - Safety Intent (SSF or Tcl)
  - Inputs of Use Case 2 or 3
- Expected outputs:
  - Modified or incremental safety intent (SSF')
  - Outputs of Use Case 2 or 3
- Related commands: load_ssf, save_ssf in use with compile/compile_ultra

For this use case of *Design Compiler*, the following conditions of use (constraints for the design and design environment, recommended procedures for the tool usage, etc.) shall be met:

- CoU- DC-001: User shall review all error and warning messages and take appropriate action.
- CoU- DC-002: User shall follow the Design Compiler Reference Methodology or use equivalent scripts.
- CoU-DC-003: For the final run, Tcl script-based batch mode execution shall be used, without interactive command line entry or GUI manual command entry. Tcl scripts and log files shall be retained as design signoff records.
- CoU-DC-004: Pre-route timing and power calculations are not considered accurate enough for final design signoff. User shall validate timing and power requirements post-route in an appropriate signoff tools (such as PrimeTime and PrimePower) prior to delivery of the final product.

For this use case of *Design Compiler*, the following assumptions of use (required actions to be taken by the tool user to prevent or detect design errors due to possible tool malfunctions) shall be met:

- AoU-DC-002: User shall check that all outputs are generated with an up-to-date timestamp.
- AoU-DC-009: User shall review synthesis log for errors, unresolved references, unmapped cells and expected results (log must include output from the following commands: check_design, report_timing, report_area, compile/compile_ultra, and check_mv_design).
- AoU-DC-011: User shall review floorplan visually in synthesis or place and route tool (such as Design Compiler NXT or IC Compiler or IC Compiler II).

- AoU-DC-022: When SSF has been utilized, user shall review the relevant safety reports (report_safety_status, report_safety_register_groups, report_fsm) and validate expected safety measures are implemented.
- AoU-DC-023: When SSF has been utilized, user shall validate correctness of any SSF output from the tool. (for example, generate safety reports; write SSF' and read it back into the tool; generate safety reports and compare).
- AoU-DC-024: When SSF has been utilized, user shall perform independent formal equivalence checking of RTL to post-synthesis netlist using a formal verification tool that supports safety intent checking (such as Formality) including SVF information and applicable safety intent. The netlist verified should be same netlist and format that is to be used in later implementation and verification steps.

All analyzed failure modes and prevention, detection and mitigation measures (including conditions and assumptions of use listed above) are independent of the exact Design Compiler tool version.

A software tool criteria evaluation performed by Synopsys according to ISO 26262-8, clause 11, which assumes the fulfillment of all conditions of use (CoU) and assumptions of use (AoU) as described above, results in a required tool confidence level:

TCL1 for *Design Compiler* Use Case 6: Failsafe Finite State Machine (FSM) Synthesis

In this case, no further activities for software tool qualification are required.

Use Case 9: Power Optimization and Clock Gating

In this use case, the user's main goal is to reduce leakage and dynamic power in the design.

Note: These settings are applied on top of use cases 2 or 3, the detection measures conditions and assumptions of use (CoU and AoU) for this use case are applied in addition to the detection measures for those use cases.

In this use case, the Design Compiler tool uses and generates the following main inputs and outputs.

- Affected Tools
    - Design Compiler
    - Design Compiler NXT
    - DC-Ultra
    - Design Compiler Graphical
    - Power Compiler

- Inputs:
    - Switching annotation (SAIF)
    - Inputs of Use Case 2 or 3
- Expected outputs:
    - Outputs from Use Case 2 or 3
- Related commands: compile, compile_ultra

For this use case of *Design Compiler*, the following conditions of use (constraints for the design and design environment, recommended procedures for the tool usage, etc.) shall be met:

- CoU- DC-001: User shall review all error and warning messages and take appropriate action.
- CoU- DC-002: User shall follow the Design Compiler Reference Methodology or use equivalent scripts.
- CoU-DC-003: For the final run, Tcl script-based batch mode execution shall be used, without interactive command line entry or GUI manual command entry. Tcl scripts and log files shall be retained as design signoff records.
- CoU-DC-004: Pre-route timing and power calculations are not considered accurate enough for final design signoff. User shall validate timing and power requirements post-route in an appropriate signoff tools (such as PrimeTime and PrimePower) prior to delivery of the final product.

For this use case of *Design Compiler*, the following assumptions of use (required actions to be taken by the tool user to prevent or detect design errors due to possible tool malfunctions) shall be met:

- AoU-DC-003: User shall verify post-synthesis netlist vs. RTL with a formal equivalence tool (such as Formality). The netlist verified should be the same netlist and format that is to be used in later implementation and verification steps.
- AoU-DC-007: User shall verify synthesis netlist timing in a signoff static timing tool (such as PrimeTime).
- AoU-DC-015: User shall verify power usage of final netlist in a signoff power analysis tool (such as PrimePower).
- AoU-DC-016: User shall review input clock_gating scripts for correctness and strategy of the implementation.
- AoU-DC-017: User shall review DFT DRC report in synthesis log (log to include DFT DRC report).
- AoU-DC-020: User shall review synthesis log for errors, unresolved references, unmapped cells and expected results (log must include output from the following commands: check_design, report_timing, report_area, compile/compile_ultra plus report_switching_activity

output for switching annotation percentages, report_power for power consumption, report_clock_gating, and report_clock_gating_check).

All analyzed failure modes and prevention, detection and mitigation measures (including conditions and assumptions of use listed above) are independent of the exact Design Compiler tool version.

A software tool criteria evaluation performed by Synopsys according to ISO 26262-8, clause 11, which assumes the fulfillment of all conditions of use (CoU) and assumptions of use (AoU) as described above, results in a required tool confidence level:

TCL1 for *Design Compiler* Use Case 6: Power Optimization and Clock Gating

In this case, no further activities for software tool qualification are required.

Use Case 10: Gate-to-Gate Optimization

In this use case, the user's main goal is to improve a netlist that has already gone through synthesis (in a tool such as Design Compiler). The input netlist is optimized for provided constraints like timing and area.

In this use case, the Design Compiler tool uses and generates the following main inputs and outputs.

- Inputs:
  - Gate level netlist (Verilog)
  - User constraints (SDC)
  - Logic Library (db)
  - TCL script (configuration, run script)
- Expected outputs:
  - Gate level netlist (Verilog)
  - Log file (.txt)
- Related commands: optimize_netlist

For this use case of *Design Compiler*, the following conditions of use (constraints for the design and design environment, recommended procedures for the tool usage, etc.) shall be met:

- CoU- DC-001: User shall review all error and warning messages and take appropriate action.
- CoU- DC-002: User shall follow the Design Compiler Reference Methodology or use equivalent scripts.

- CoU-DC-003: For the final run, Tcl script-based batch mode execution shall be used, without interactive command line entry or GUI manual command entry. Tcl scripts and log files shall be retained as design signoff records.
- CoU-DC-004: Pre-route timing and power calculations are not considered accurate enough for final design signoff. User shall validate timing and power requirements post-route in an appropriate signoff tools (such as PrimeTime and PrimePower) prior to delivery of the final product.

For this use case of *Design Compiler*, the following assumptions of use (required actions to be taken by the tool user to prevent or detect design errors due to possible tool malfunctions) shall be met:

- AoU-DC-002: User shall check that all outputs are generated with an up-to-date timestamp.
- AoU-DC-003: User shall verify post-synthesis netlist vs. RTL with a formal equivalence tool (such as Formality). The netlist verified should be the same netlist and format that is to be used in later implementation and verification steps.
- AoU-DC-007: User shall verify synthesis netlist timing in a signoff static timing tool (such as PrimeTime).
- AoU-DC-013: User shall review log of clock domain crossing checking tools (such as SpyGlass CDC).
- AoU-DC-021: User shall review synthesis log for errors, unresolved references, unmapped cells and expected results (log must include output from the following commands: check_design, report_timing, report_area, optimize_nelist, and report_reference).

All analyzed failure modes and prevention, detection and mitigation measures (including conditions and assumptions of use listed above) are independent of the exact Design Compiler tool version.

A software tool criteria evaluation performed by Synopsys according to ISO 26262-8, clause 11, which assumes the fulfillment of all conditions of use (CoU) and assumptions of use (AoU) as described above, results in a required tool confidence level:

TCL1 for *Design Compiler* Use Case 7: Gate-to-gate Optimization

In this case, no further activities for software tool qualification are required.

7
Limitations of Use Cases

This section describes all known limitations of the use cases mentioned in the previous section.

All known safety-related issues for the Design Compiler tool are listed in the Design Compiler Safety-Related Issues Master List available on SolvNetPlus.

LIM-1: LCA Features

Each release of the Design Compiler tool may contain hidden, undocumented features for testing or evaluation purposes, known as “Limited Customer Availability” (LCA) features. Use LCA features only for testing and evaluating the proposed new features, not for production work.

LIM-DC-1: Asynchronous Designs

Design Compiler is a synchronous design tool and as such, does not maintain the priority structure of logic in a combinational tree. This can lead to functional errors if the design is an asynchronous design.

For asynchronous design elements (not including asynchronous set/resets) it is recommended that users instantiate the logic structures directly in the RTL and do not allow the Design Compiler to re-structure via constraints such as size_only.

LIM-DC-2: Multi-Voltage Design without UPF

Designs with shutdown regions (power domains) can be implemented outside of the UPF design flow. For instance, it is possible to insert all isolation cells into the design by hand. If this approach is taken, Formality cannot validate the design behavior during shutdown and there will be a gap in the validation.

For safety-related designs with shutdown regions UPF should be used.

LIM-DC-3: User Instantiated DesignWare (including building blocks)

This limitation applies to any user instantiated instances where the RTL functionality was written by a third party other than the user. Examples of this could be DesignWare elements or third-party IP. Only the connectivity of these elements can be verified by the Design Compiler tool chain. The functionality of these elements cannot be validated by the tool chain and must be validated by other means.

Building block DesignWare: DesignWare elements that are automatically inferred by Design Compiler from the user RTL are covered by the validation tool chain. The functionality of the gates from the DesignWare part will be verified vs the user RTL description. However, if the parts are instantiated instead of inferred, there is no user RTL to compare against and the correctness of the DesignWare building block will have to be evaluated by other means.

LIM-DC-4: Random Access Memories (RAMs)

RAMs are user instantiated cells that generally have a different underlying physical structure that is more vulnerable to SEUs than a typical cell. For guidelines on picking fault-tolerant RAM implementations to instantiate, consult with the vendor of your manufacturing technology.

LIM-DC-5: Safe State Machine Synthesis

The following method described applies to versions of the Design Compiler tool prior to P-2019.03-SP4. Staring with P-2019.03-S4, there is an integrated and supported tool feature set for Safe State Machine synthesis and re-encodingFor information, see the tool documentation.

For the purposes of this document, a safe state machine is defined as one that will return to a known state after a SEU.

This article describes how to model the safe state machine in RTL and configure Design Compiler (DC) such that

- The functional states have pre-defined names for ease of reading
- Known behavior the non-functional state is preserved through synthesis

Known behavior the non-functional state is preserved through synthesis RTL Coding Style

There are generally several ways to model state machine logic in RTL. This article focuses on where the sequential state registers are in one process and the combinational next state logic is in a separate process. This is the generally recommended coding style and tends to be easier to read.

State Encoding

There are a couple of methods available in VHDL for specifying the functional states of a state machine.

The first is to create type with enumerated values matching the functional states.

SystemVerilog

```
  typedef enum {s0, s1, s2, s3, s4,s5} state_values;
  state_values state, next_state;
```

VHDL

```
type state_values is (s0,s1,s2,s3,s4,s5);
signal state, next_state: state_values;
```

The second is to create named constants with the state encoding

SystemVerilog

```
localparam logic [5:0] s0 = 6’b000001;
localparam logic [5:0] s1 = 6’b000010;
localparam logic [5:0] s2 = 6’b000100;
localparam logic [5:0] s3 = 6’b001000;
localparam logic [5:0] s4 = 6’b010000;
localparam logic [5:0] s5 = 6’b100000;
```

VHDL

```
constant S0 : std_logic_vector(5 downto 0) := B"000001";
constant S1 : std_logic_vector(5 downto 0) := B"000010";
constant S2 : std_logic_vector(5 downto 0) := B"000100";
constant S3 : std_logic_vector(5 downto 0) := B"001000";
constant S4 : std_logic_vector(5 downto 0) := B"010000";
constant S5 : std_logic_vector(5 downto 0) := B"100000";
```

For the purposes of synthesis in DC the second style is required. When an enumerated type is specified, DC will treat values outside of the enumeration and don’t_cares. This negates our ability to specify a known behavior for non-enumerated state with the others clause (as we’ll see later in the article)

To ensure that a SEU doesn’t cause the state machine to jump to another random functional state, users should specify a state encoding that allows detection of a SEU like one-hot or Hamming-2

Specifying Behavior of non-Functional States

Most often the next state logic for a state machine is specified in a case statement. Using the named constants above allows a readable format for the functional states.

SystemVerilog

```
case (state)
  s0 : begin
    out1 <= “00”;
    next_state <= in1 ? s0 : s1;
  end
  s1 : begin
    out1 <= “01”;
    next_state <= s2;
  end
  s2 : begin
    out1 <= “10”;
    next_state <= s3;
  end
```

```
    s3 : begin
      out1 <= “10”;
      next_state <= s4;
    end
    s4 : begin
      out1 <= “11”;
      next_state <= s5;
    end
    s5 : begin
      out1 <= “11”;
      next_state <= s0;
    end
    default : begin
      out1 <= “00”;
      next_state <= s0;
    end
```

VHDL

```
  case state is
    when s0 => out1 <="00";
      if in1 = '0' then
        next_state <= s0;
      else
        next_state <= s1;
      end if;
    when s1 => out1 <="01";
      next_state <= s2;
    when s2 => out1 <="10";
      next_state <= s3;
    when s3 => out1 <="10";
      next_state <= s4;
    when s4 => out1 <="11";
      next_state <= s5;
    when s5 =>  out1  <= "11";
      next_state <= s0;
    when others => out1  <= "11";
      next_state <= s0;
```

The “default” or “others” clause will catch all the non-functional states and synthesis will create logic to return the state machine to the specified state on the next clock. As mentioned earlier if an enumerated type is used for the state vector, DC will consider the others clause unreachable and will synthesize don’t-care logic for the unreachable states.

Note: If the desire is for the state machine to return to the reset state in the event of an upset, it is the users responsibility to ensure the state specified in the others clause matches the state specified in the reset condition of the state registers.

Automatic State Encoding during compile_ultra

Design Compiler can re-encode state machines with different encoding schemes (binary, grey, onehot). However, this capability is limited to state machines specified with enumerated types and cannot handle non-functional states. As such, it is not useful for fault tolerant state machines.

Requirements

By using the following coding styles

- State encoding using constants
- Other clauses to specify behavior for non-functional state

Users can ensure that Design Compiler synthesizes a state machine with known behavior for non-functional states.

Appendix A
Software Tool Information

This section provides general information about the Design Compiler software tool, which is needed by the tool user for performing his/her software tool criteria evaluation.

The following information about Design Compiler is required according to ISO 26262-8, for the planning of the usage of a software tool (clause 11.4.4) and the preparation of the own software tool criteria evaluation (clause 11.4.5).

Please note that some of the information below provided by Synopsys simply needs to be confirmed by the tool user and can be used without modification. Other information must be completed or updated by the tool user to reflect his/her actual situation.

Required Info	Tool Information	Reference / Comment
Tool vendor	Synopsys, Inc.	ISO 26262-8, 11.4.4.1.a
Tool name and version	Design Compiler DC Ultra Design Compiler Graphical Design Compiler NXT HDL Compiler Power Compiler N-2017.09 and later versions	ISO 26262-8, 11.4.4.1.a To determine tool version, use: `report_version -options`
Tool use cases		ISO 26262-8, 11.4.4.1.c ISO 26262-8, 11.4.5.1.a To be completed by the tool user. Align with / verify against use cases described in Section 6 of this document.
Tool inputs and expected outputs		ISO 26262-8, 11.4.5.1.b To be completed by the tool user. Align with / verify against inputs and outputs described in Section 6 of this document.
Tool configuration and constraints		ISO 26262-8, 11.4.4.1.b ISO 26262-8, 11.4.5.1.c To be completed by the tool user. Align with / verify against CoU for the use cases described in Section 6 of this document.
Tool environment (OS)	Refer to the Synthesis Installation Notes at https://www.synopsys.com/install.	ISO 26262-8, 11.4.4.1.d

	Click the Design Compiler tool name, the release number, and then "View installation guide" for tool version-specific OS support.	To be completed by the tool user. Align with / verify against the OS version evaluated by Synopsys. To determine Linux version, use: `uname -osr`
Tool environment (CAD tool chain)		ISO 26262-8, 11.4.4.1.d To be completed by the tool user. To determine name and version of your tool chain, please consult your CAD department.
Maximum ASIL	ASIL D	ISO 26262-8, 11.4.4.1.e
Tool qualification methods	Not applicable	ISO 26262-8, 11.4.4.1.f Software tool qualification is not required for Design Compiler
User manual and other usage guide documents	See Design Compiler Online Help and Reference Methodology on SolvNetPlus.	ISO 26262-8, 11.4.4.2.a – d Tool user to include a link to these documents (Synopsys SolvNetPlus or local copy), and to add any additional company-internal tool usage guidelines.
Known software tool malfunctions, and appropriate work arounds ...	For limitations, refer to Section 7 of this document. For the published list of safety-related defects for the Design Compiler tool, see Design Compiler Safety-Related Issues Master List on SolvNetPlus	ISO 26262-8, 11.4.4.2.e Tool user to include a link to these documents (Synopsys SolvNetPlus or local copy), and to add any additional company-internal work around descriptions.
Measures for the detection of tool malfunctions ...		ISO 26262-8, 11.4.4.2.f To be completed by the tool user. Align with / verify against AoU for the use cases described in Section 6 of this document.

Appendix B
Complete List of CoU and AoU IDs

The complete list of Conditions of Use (CoU) for Design Compiler is shown in the table below. CoU define a condition of the design, software tool, design environment, or situation that is assumed and required to be fulfilled by the user.

ID	Description
CoU-DC-001	User shall review all error and warning messages and take appropriate action.
CoU-DC-002	User shall follow the Design Compiler Reference Methodology or use equivalent scripts.
CoU-DC-003	For the final run, Tcl script-based batch mode execution shall be used, without interactive command line entry or GUI manual command entry. Tcl scripts and log files shall be retained as design signoff records.
CoU-DC-004	Pre-route timing and power calculations are not considered accurate enough for final design signoff. User shall validate timing and power requirements post-route in an appropriate signoff tools (such as PrimeTime and PrimePower) prior to delivery of the final product.
CoU-DC-005	User shall run load_upf prior to issuing the compile/compile_ultra command.
CoU-DC-006	User shall follow the Design Compiler NXT Reference Methodology including the link to IC Compiler II or use equivalent scripts.
CoU-ICCII-001*	User shall review all error and warning messages and take appropriate action.

The complete list of Assumptions of Use (AoU) for Design Compiler is shown in the table below. AoU define an action that is assumed and required to be taken by the user of a software tool.

ID	Description
AoU-DC-001	User shall review synthesis log for errors, unresolved references, unmapped cells and expected results (log must include elaboration report).
AoU-DC-002	User shall check that all outputs are generated with an up-to-date timestamp.
AoU-DC-003	User shall verify post-synthesis netlist vs. RTL with a formal equivalence tool (such as Formality). The netlist verified should be the same netlist and format that is to be used in later implementation and verification steps.

ID	Description
AoU-DC-004	User shall review synthesis log for errors, unresolved references, unmapped cells and expected results (log must include output from the following commands: check_design, report_timing, report_area, compile/compile_ultra, and report_reference).
AoU-DC-005	User shall review paths with infer_mux in GTECH netlist to ensure MUX_OP is used for the selector.
AoU-DC-006	User shall verify synthesis or place and route netlist by gate-level simulation for correct reset behavior using a logic simulator (such as VCS).
AoU-DC-007	User shall verify synthesis netlist timing in a signoff static timing tool (such as PrimeTime).
AoU-DC-008	User shall review logs of DFT tool (such as DFT Compiler or TestMAX DFT) for errors and verify scan coverage.
AoU-DC-009	User shall review synthesis log for errors, unresolved references, unmapped cells and expected results (log must include output from the following commands: check_design, report_timing, report_area, compile/compile_ultra, and check_mv_design).
AoU-DC-010	User shall review paths with infer_mux in synthesis netlist to ensure mux cells are used.
AoU-DC-011	User shall review floorplan visually in synthesis or place and route tool (such as Design Compiler NXT or IC Compiler or IC Compiler II).
AoU-DC-012	User shall review place and route tool (such as IC Compiler or IC Compiler II) log file for errors during read (place and route is the next implementation step in the digital tool chain).
AoU-DC-013	User shall review log of clock domain crossing checking tools (such as SpyGlass CDC).
AoU-DC-014	User shall review low power static checking tool results (such as VC LP) run on gate-level netlist plus UPF'. This will compare original intent vs UPF' output intent.
AoU-DC-015	User shall verify power usage of final netlist in a signoff power analysis tool (such as PrimePower).
AoU-DC-016	User shall review input clock_gating scripts for correctness and strategy of the implementation.

ID	Description
AoU-DC-017	User shall review DFT DRC report in synthesis log (log to include DFT DRC report).
AoU-DC-018	User shall verify post-synthesis netlist vs. RTL with a formal equivalence tool (such as Formality) including UPF verification. The netlist verified should be same netlist and format that is to be used in later implementation and verification steps.
AoU-DC-019	User shall verify synthesis netlist timing in a signoff static timing tool (such as PrimeTime) including voltage violations.
AoU-DC-020	User shall review synthesis log for errors, unresolved references, unmapped cells and expected results (log must include output from the following commands: check_design, report_timing, report_area, compile/compile_ultra plus report_switching_activity output for switching annotation percentages, report_power for power consumption, report_clock_gating, and report_clock_gating_check).
AoU-DC-021	User shall review synthesis log for errors, unresolved references, unmapped cells and expected results (log must include output from the following commands: check_design, report_timing, report_area, optimize_nelist, and report_reference).
AoU-DC-022	When SSF has been utilized, user shall review the relevant safety reports (report_safety_status, report_safety_register_groups, report_fsm) and validate expected safety measures are implemented.
AoU-DC-023	When SSF has been utilized, user shall validate correctness of any SSF output from the tool. (for example, generate safety reports; write SSF' and read it back into the tool; generate safety reports and compare).
AoU-DC-024	When SSF has been utilized, user shall perform independent formal equivalence checking of RTL to post-synthesis netlist using a formal verification tool that supports safety intent checking (such as Formality) including SVF information and applicable safety intent. The netlist verified should be same netlist and format that is to be used in later implementation and verification steps.
AoU-DC-025	User shall verify gate-level SPFM meets design ASIL requirement on post-synthesis netlist with a fault modeling tool (such as TestMAX FuSa or Z01X). The netlist verified should be same netlist and format that is to be used in later implementation and verification steps.
AoU-ICCII-001*	User shall review the log files for error messages, warning messages, correct, complete, or unexpected flow execution, and random characters.
AoU-ICCII-002*	User shall review all relevant QoR reports, such as timing and power reports, for error messages, warning messages, random characters, and expected results.

ID	Description
AoU-ICCII-004*	User shall perform independent formal equivalence checking of the gate-level netlist before and after place and route using a formal verification tool (such as Formality).
AoU-ICCII-022*	User shall review the log files for error messages, warning messages, and correct operation during the saving and subsequent reloading of the NDM libraries.

*IC Compiler II AoUs / CoUs are applicable only in combination with Design Compiler NXT and Use Case 4.

Appendix C
List of CoU and AoU Changes

Assumptions of Use (AoU) for Design Compiler which has changed along with a description of that change when compared to a previous document version indicated are shown in the table below.

ID	Description of Change	Version
AoU-DC-022	Added new AoU: When SSF has been utilized, user shall review the relevant safety reports (report_safety_status, report_safety_register_groups, report_fsm) and validate expected safety measures are implemented.	v2.0
AoU-DC-023	Added new AoU: When SSF has been utilized, user shall validate correctness of any SSF output from the tool. (for example, generate safety reports; write SSF' and read it back into the tool; generate safety reports and compare).	v2.0
AoU-DC-024	Added new AoU: When SSF has been utilized, user shall perform independent formal equivalence checking of RTL to post-synthesis netlist using a formal verification tool that supports safety intent checking (such as Formality) including SVF information and applicable safety intent. The netlist verified should be same netlist and format that is to be used in later implementation and verification steps.	v2.0
AoU-DC-025	Added new AoU: User shall verify gate-level SPFM meets design ASIL requirement on post-synthesis netlist with a fault modeling tool (such as TestMAX FuSa or Z01X). The netlist verified should be same netlist and format that is to be used in later implementation and verification steps.	v2.0