cādence®

# Tutorial: Virtuoso ADE Verifier Workflow

**Product Version ICADVM20.1**
**October 2020**

# Contents

# Preface

This document provides information about using Virtuoso ADE Verifier (Verifier) for requirements-driven analog design verification flows.

This document is intended for project managers, verification leaders, and analog designers who perform requirements-driven verification of analog designs. Readers should be familiar with the applications used to simulate and analyze analog designs in the Virtuoso design environment, notably Virtuoso ADE Assembler and Virtuoso ADE Explorer.

This preface contains the following topics:

## Scope

Unless otherwise noted, the functionality described in this guide can be used in both mature node (for example, IC6.1.8) and advanced node (for example, ICADVM20.1) releases.

| Label | Meaning |
|---|---|
| **(ICADVM20.1 Only)** | Features supported only in ICADVM20.1 advanced nodes and advanced methodologies releases. |
| **(IC6.1.8 Only)** | Features supported only in mature node releases. |

## Licensing Requirements

Verifier is available with the license 95270 (Virtuoso_ADE_Verifier) in the IC6.1.7 and higher releases.

Appropriate licenses are required to simulate implementations from Verifier. These include the licenses for Virtuoso ADE Assembler, Virtuoso ADE Explorer, and the appropriate simulator licenses for simulating designs and using the appropriate simulation environment.

## Related Documentation

### What's New and KPNS

- *Virtuoso ADE Verifier What's New*
- *Virtuoso ADE Verifier Known Problems and Solutions*

### Installation, Environment, and Infrastructure

- *Virtuoso ADE Verifier User Guide*
- *Virtuoso ADE Verifier SKILL Reference*

### Technology Information

- *Virtuoso Technology Data User Guide*
- *Virtuoso Technology Data ASCII Files Reference*
- *Virtuoso Technology Data SKILL Reference*

### Virtuoso Tools

- *Virtuoso ADE Verifier User Guide*
- *Tutorial: Virtuoso ADE Verifier Workflow*
- *Virtuoso ADE Assembler User Guide*
- *Virtuoso ADE Explorer User Guide*
- *Virtuoso Schematic Editor User Guide*

- *Virtuoso UltraSim Simulator User Guide*
- *Spectre AMS Designer Simulator User Guide*
- *Spectre Circuit Simulator and Accelerated Parallel Simulator RF Analysis in ADE Explorer User Guide*
- *Spectre Circuit Simulator and Accelerated Parallel Simulator User Guide*
- *Spectre Circuit Simulator Reference*

## Additional Learning Resources

### Video Library

The Video Library on the Cadence Online Support website provides a comprehensive list of videos on various Cadence products.

To view a list of videos related to a specific product, you can use the *Filter Results* feature available in the pane on the left. For example, click the *Virtuoso Layout Suite* product link to view a list of videos available for the product.

You can also save your product preferences in the Product Selection form, which opens when you click the *Edit* icon located next to *My Products*.

### Virtuoso Videos Book

You can access certain videos directly from Cadence Help. To learn more about this feature and to access the list of available videos, see Virtuoso Videos.

### Rapid Adoption Kits

Cadence provides a number of Rapid Adoption Kits that demonstrate how to use Virtuoso applications in your design flows. These kits contain design databases and instructions on how to run the design flow.

Virtuoso ADE Verifier Rapid Adoption Kit

In addition, Cadence offers the following training courses on ADE Verifier:

- Virtuoso ADE Verifier

- Virtuoso ADE Explorer S1: Set Up and Run Analog Simulations Using the Spectre Simulator
- Virtuoso ADE Explorer S2: Analyzing Simulations Using the ViVA XL Waveform Tool
- Virtuoso ADE Explorer S3: Corner Analysis and Monte Carlo Simulations
- Virtuoso ADE Explorer S4: Real-Time Tuning, Checks/Asserts, and Reliability Analysis
- Virtuoso ADE Assembler S1: Introducing the Assembler Environment
- Virtuoso ADE Assembler S2: Sweeping Variables, Simulating Corners, and Creating Run Plans
- Virtuoso ADE Assembler S3: Circuit Checks, Device Asserts, and Reliability Analysis
- Variation Analysis Using the Virtuoso ADE Assembler

To explore the full range of training courses provided by Cadence in your region, visit Cadence Training or write to training_enroll@cadence.com.

**Note:** The links in this section open in a separate web browser window when clicked in Cadence Help.

## Help and Support Facilities

Virtuoso offers several built-in features to let you access help and support directly from the software.

- The Virtuoso *Help* menu provides consistent help system access across Virtuoso tools and applications. The standard Virtuoso *Help* menu lets you access the most useful help and support resources from the Cadence support and corporate websites directly from the CIW or any Virtuoso application.
- The Virtuoso Welcome Page is a self-help launch pad offering access to a host of useful knowledge resources, including quick links to content available within the Virtuoso installation as well as to other popular online content.

  The Welcome Page is displayed by default when you open Cadence Help in standalone mode from a Virtuoso installation. You can also access it at any time by selecting *Help – Virtuoso Documentation Library* from any application window, or by clicking the *Home* button on the Cadence Help toolbar (provided you have not set a custom home page).

For more information, see Getting Help in *Virtuoso Design Environment User Guide.*

## Customer Support

For assistance with Cadence products:

- Contact Cadence Customer Support

  Cadence is committed to keeping your design teams productive by providing answers to technical questions and to any queries about the latest software updates and training needs. For more information, visit https://www.cadence.com/support.

- Log on to Cadence Online Support

  Customers with a maintenance contract with Cadence can obtain the latest information about various tools at https://support.cadence.com.

## Feedback about Documentation

You can contact Cadence Customer Support to open a service request if you:

- Find erroneous information in a product manual
- Cannot find in a product manual the information you are looking for
- Face an issue while accessing documentation by using Cadence Help

You can also submit feedback by using the following methods:

- In the Cadence Help window, click the *Feedback* button and follow instructions.
- On the Cadence Online Support Product Manuals page, select the required product and submit your feedback by using the *Provide Feedback* box.

## Typographic and Syntax Conventions

The following typographic and syntax conventions are used in this manual.

| | |
|---|---|
| *text* | Indicates names of manuals, menu commands, buttons, and fields. |
| `text` | Indicates text that you must type as presented. Typically used to denote command, function, routine, or argument names that must be typed literally. |
| *`z_argument`* | Indicates text that you must replace with an appropriate argument value. The prefix (in this example, *`z_`*) indicates the data type the argument can accept and must not be typed. |
| \| | Separates a choice of options. |
| `{ }` | Encloses a list of choices, separated by vertical bars, from which you **must** choose one. |
| `[ ]` | Encloses an optional argument or a list of choices separated by vertical bars, from which you **may** choose one. |
| `[ ?argName` *`t_arg`* `]` | Denotes a *key argument*. The question mark and argument name must be typed as they appear in the syntax and must be followed by the required value for that argument. |
| `...` | Indicates that you can repeat the previous argument.<br>Used with brackets to indicate that you can specify zero or more arguments.<br>Used without brackets to indicate that you must specify at least one argument. |
| `,...` | Indicates that multiple arguments must be separated by commas. |
| `=>` | Indicates the values returned by a Cadence SKILL language function. |
| `/` | Separates the values that can be returned by a Cadence SKILL language function. |

If a command-line or SKILL expression is too long to fit within the paragraph margins of this document, the remainder of the expression is moved to the next line and indented. In code excerpts, a backslash ( \ ) indicates that the current line continues on to the next line.

# 1 Getting Started

This chapter introduces Virtuoso ADE Verifier (Verifier). It includes the following topics:

- Introducing Verifier
- Using Verifier in Design Flows
- About the Sample Design Used in this Verifier Tutorial

Video

For a video overview of Verifier, see *Introducing Virtuoso ADE Verifier* on Cadence Online Support.

*RAK*

You can download the Virtuoso ADE Verifier Rapid Adoption Kit from Cadence Online Support. This kit also contains the sample design database and instructions on how to implement the basic verification flow for a design. This sample design database is also packaged with this tutorial. See “Obtaining the Sample Design”.

## Introducing Verifier

Verifier is a comprehensive application to conduct and monitor requirements-driven verification flows for analog designs. You can use Verifier to meet the following objectives for your design verification flows:

- Define the verification requirements for a design project.
- Map the verification requirements with their implementations, which can be ADE Assembler and ADE Explorer cellviews of the type `maestro`, their tests, and outputs.
- Set owners of requirements when multiple designers are involved in the project.
- Simulate multiple implementation cellviews individually, serially, or in parallel.
- Perform and monitor large batch-based verification regression runs.
- Monitor the requirements-based verification status of the design project through various reports.

Verifier maintains verification data in cellviews of the type `verifier` that contain:

- The verification plan, structured as a hierarchy of verification requirements
- Implementation links and the mapping between requirements and their corresponding implementations
- Verifier preferences, such as the default job policy to use, the source for verification specifications that Verifier uses to determine verification results, among other settings

Verifier also lets you create owner cellviews of the type `verifier` based on the owners of the requirements. This feature helps project managers assign owners to design verification requirements and monitor the verification status of the owned requirements. Requirement owners can create implementations and map them to their requirements.

Verifier also provides advanced features, such as the ability to add your own custom fields. It also offers a configurable environment and full data access through an extensive set of SKILL functions. For details on these functions, see *Virtuoso ADE Verifier SKILL Reference*.

The following table describes the key elements of Verifier.

| **Element** | **Description** |
|---|---|
| Requirements | A design verification plan includes a set of verification requirements structured in a hierarchical tree. A verification requirement is an abstract description of an aspect of a design that needs to be verified. It includes details like the title, description, and verification goal with the applicable range of acceptable output values. In a multi-user design environment, requirements can have owners. You can also specify custom fields to include additional information. |
| Implementations | An implementation represents the simulation setup, testbench, or the measured output of a requirement. Implementation data is stored in cellviews of the type `maestro`. You map requirements to the corresponding implementations. To determine the status of a design verification requirement, you simulate the associated implementation. Verifier compares the outputs with the specification values set in the related implementations or requirement and determines the verification results.<br><br>**Note:** Verifier supports the checks and asserts defined in cellviews as implementations. |
| Verification status and reports | Verifier presents the overall requirements-based design verification status and detailed verification results in its user interface. You can also generate reports in HTML format. |

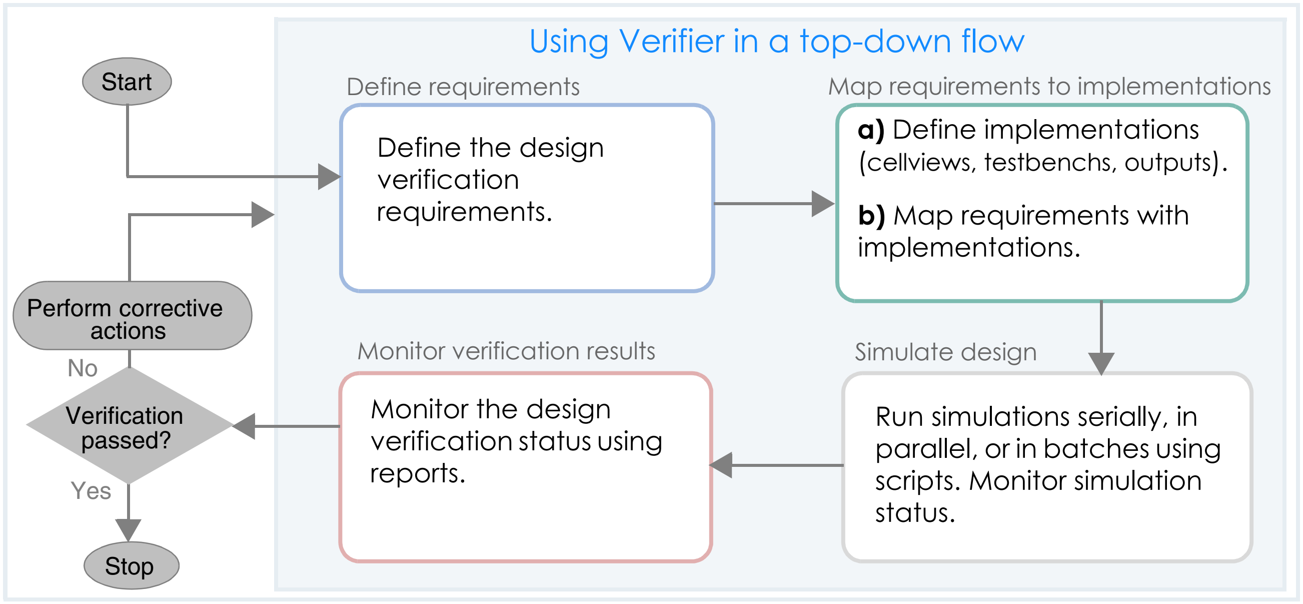
## Using Verifier in Design Flows

You can customize the use of Verifier according to your design verification needs. For example, you can use Verifier in a top-down flow where you define requirements before you add their implementations. Alternatively, you can use Verifier in a bottom-up flow where you add implementations before you start developing the verification plan in the form of requirements. You can also use Verifier in a meet-in-the-middle flow that has a mix of the top-down and bottom-up elements.

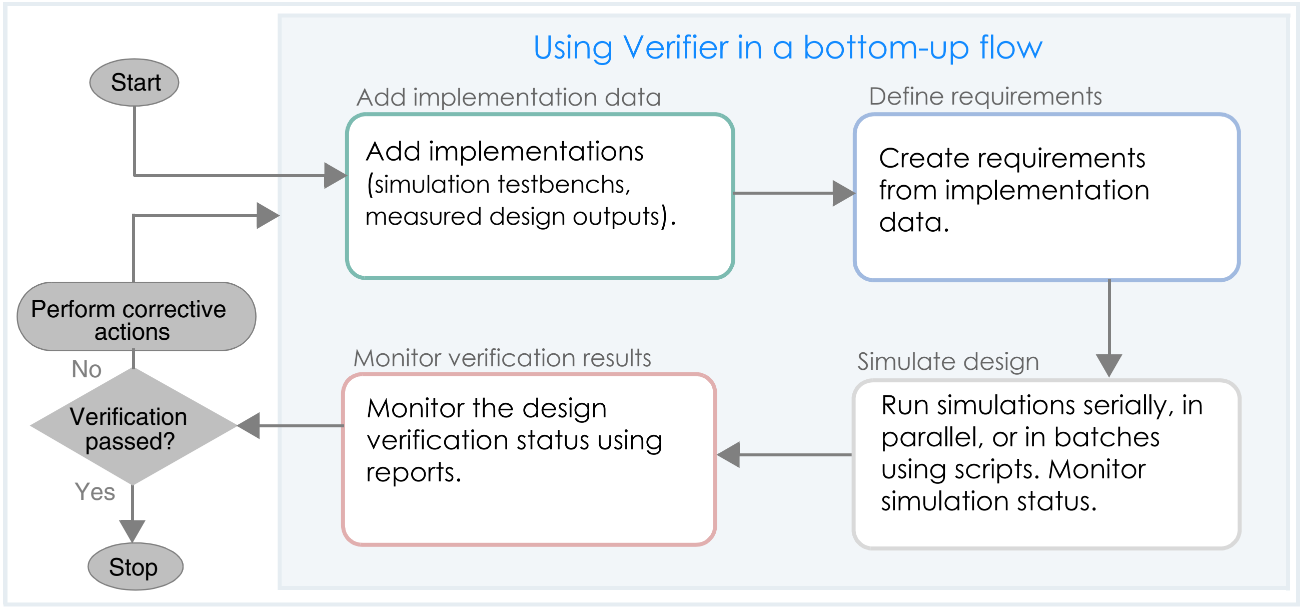
**Note:** The top-down flow is recommended when you want to reuse a verification plan in different projects. Use the bottom-up flow if the design project is in progress and implementations already exist.

When you launch a new Verifier setup, Verifier prompts you to include implementations for a bottom-up flow, include requirements for a top-down flow, or start Verifier without any data.

The following figure, with links to relevant sections, illustrates how you can use Verifier in a generic top-down verification flow.


Using Verifier in a top-down flow
Start
Define requirements
Define the design verification requirements.
Map requirements to implementations
**a)** Define implementations (cellviews, testbenches, outputs).
**b)** Map requirements with implementations.
Simulate design
Run simulations serially, in parallel, or in batches using scripts. Monitor simulation status.
Monitor verification results
Monitor the design verification status using reports.
Verification passed?
No
Perform corrective actions
Yes
Stop

The following figure, with links to relevant sections, illustrates how you can use Verifier in a generic bottom-up verification flow.


Using Verifier in a bottom-up flow
Start
Add implementation data
Add implementations (simulation testbenches, measured design outputs).
Define requirements
Create requirements from implementation data.
Simulate design
Run simulations serially, in parallel, or in batches using scripts. Monitor simulation status.
Monitor verification results
Monitor the design verification status using reports.
Verification passed?
No
Perform corrective actions
Yes
Stop

## About the Sample Design Used in this Verifier Tutorial

This section describes the following topics:

- Introducing the Sample Design
- Obtaining the Sample Design

### Introducing the Sample Design

This tutorial demonstrates how you can use Verifier to perform design verification tasks for a sample design. This sample design is part of the unified Rapid Adopt Kit (RAK) database for ADE Explorer, ADE Assembler, and ADE Verifier.

The verification project for this design is called Analog Verification Project (AVP). This tutorial illustrates a simple top-down verification flow for the sample design. It also illustrates how you use various advanced features of Verifier for the sample design.

*Tip*

You can use the top-down and bottom-up verification flow diagrams in this document to learn about employing Verifier in these flows. These diagrams contain links to the topics where you can find details on the related tasks.

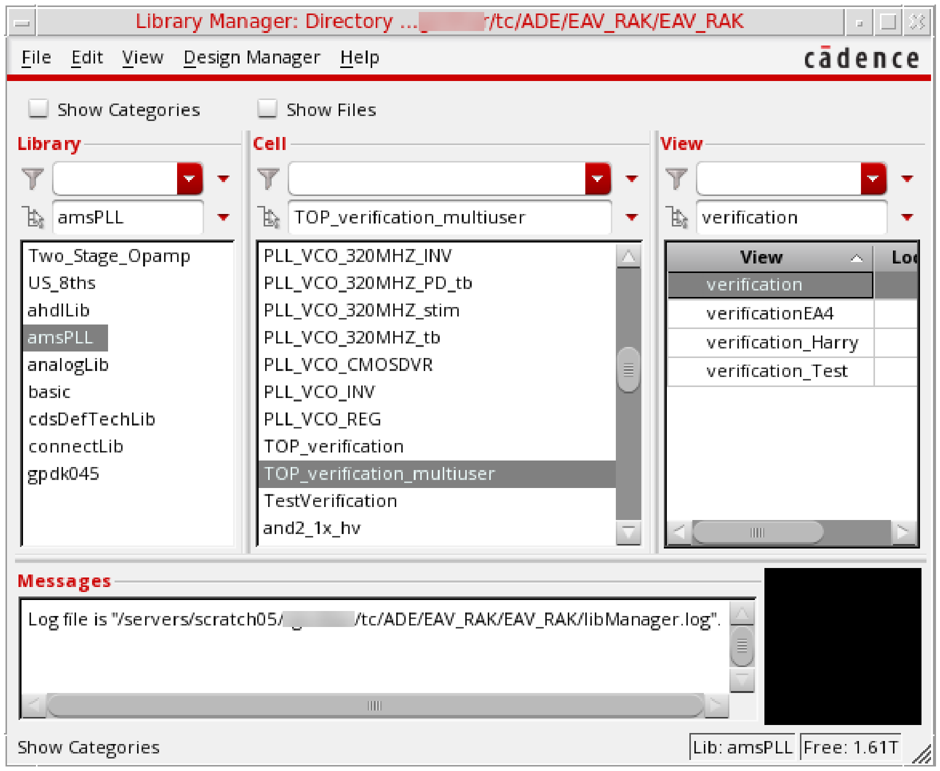
The sample design consists of the following blocks:

- OpAmp
- Filter
- PowerAmp
- VCO

This tutorial assumes that the current status of the sample design project is as follows:

- Different designers and verification engineers are working on different blocks in this design project.
- The design and verification cycle of the OpAmp and VCO blocks is in progress.
- The design and verification cycle of the Filter and PowerAmp blocks is planned for the future.

The following figure illustrates the library structure of the sample design.



Log file is "/servers/scratch05/[redacted]/tc/ADE/EAV_RAK/EAV_RAK/libManager.log".

Note the following:

- The VCO block data is stored in the library `amsPLL` and the OpAmp block data is stored in the library `Two_Stage_Opamp`. This tutorial uses these libraries to demonstrate the basic verification flow performed using Verifier.
- The libraries `amsPLL` contain cellviews of the type `verifier` that is used to demonstrate how you use Verifier.

## Obtaining the Sample Design

The `.tar.gz` file that contains the sample design is packaged with this tutorial. Download and extract this file to a location from where you want to use Verifier. See the following procedure for details. The procedure described below assumes that the filename is `EAV_RAK.tar.gz`.

**Note:** The Virtuoso ADE Verifier Rapid Adoption Kit on Cadence Online Support also contains the sample design database and instructions on how to implement the basic verification flow for a design.

## To obtain and prepare the sample design database for exploring Verifier

1. Copy the `.tar.gz` file from the `dbSrc` directory of this tutorial to your destination directory from where you want to explore Verifier.

   For example:

   ```
   $ cp /Virtuoso_Installation_Dir/doc/verifierWorkflow/dbSrc/
   EAV_RAK.tar.gz /myHome
   ```

   **Note:** If you are accessing this tutorial from Cadence Help, right-click the page and select *Copy Current Location*. Use the copied text to identify the location of this tutorial and its `dbSrc` directory.

2. Change to the destination directory.

   For example:

   ```
   $ cd /myHome
   ```

3. Extract the contents of the tar file containing the design database.

   For example:

   ```
   $ tar -xvf EAV_RAK.tar.gz
   ```

4. Change to the directory containing the extracted sample design database.

   For example, if the directory created by extracting the `.tar.gz` file is `EAV_RAK`, change to this directory.

   ```
   $ cd EAV_RAK
   ```

5. Start Virtuoso after ensuring that the Virtuoso and MMSIM paths are set correctly.

   ```
   $ virtuoso &
   ```

You can now use Library Manager to access the design database and start exploring Verifier by performing the tasks described in the subsequent chapters. For more information on the sample design, contact your Cadence representative.

# 2 Performing Basic Verification Flow Tasks

This chapter demonstrates how Virtuoso ADE Verifier (Verifier) is used to perform a basic design verification flow. This basic verification flow, in the top-down verification sequence, is demonstrated using the existing data in the sample design.

This chapter includes the following topics:

**Notes:**

- You can use Verifier in different verification flows, and not just the top-down flow demonstrated in this chapter.
- This chapter showcases selected features of Verifier that are typically used to perform basic design verification. For details about other features, see Chapter 3, "Using Advanced Verifier Features."

## Defining Verification Requirements

This section details the following tasks.

- Task 1: Open a Verifier Cellview
- Task 2: Review the Requirement Hierarchy
- Task 3: Edit a Requirement
- Task 4: Add and Move a Requirement
- Task 5: Delete a Requirement

### Task 1: Open a Verifier Cellview

Open `amsPLL/TOP_verification/verification` in Verifier. This cellview contains the design verification plan of the sample design.

**To open the cellview**

1. Choose *Tools — Library Manager* from Virtuoso CIW to launch Virtuoso Library Manager.
2. Double-click the cellview of the type `verifier`.

   For this task, select `amsPLL/TOP_verification/verification`. Double-click `verification`. Virtuoso launches Verifier and displays the data stored in this view. *Setup* is the default tab of Verifier.

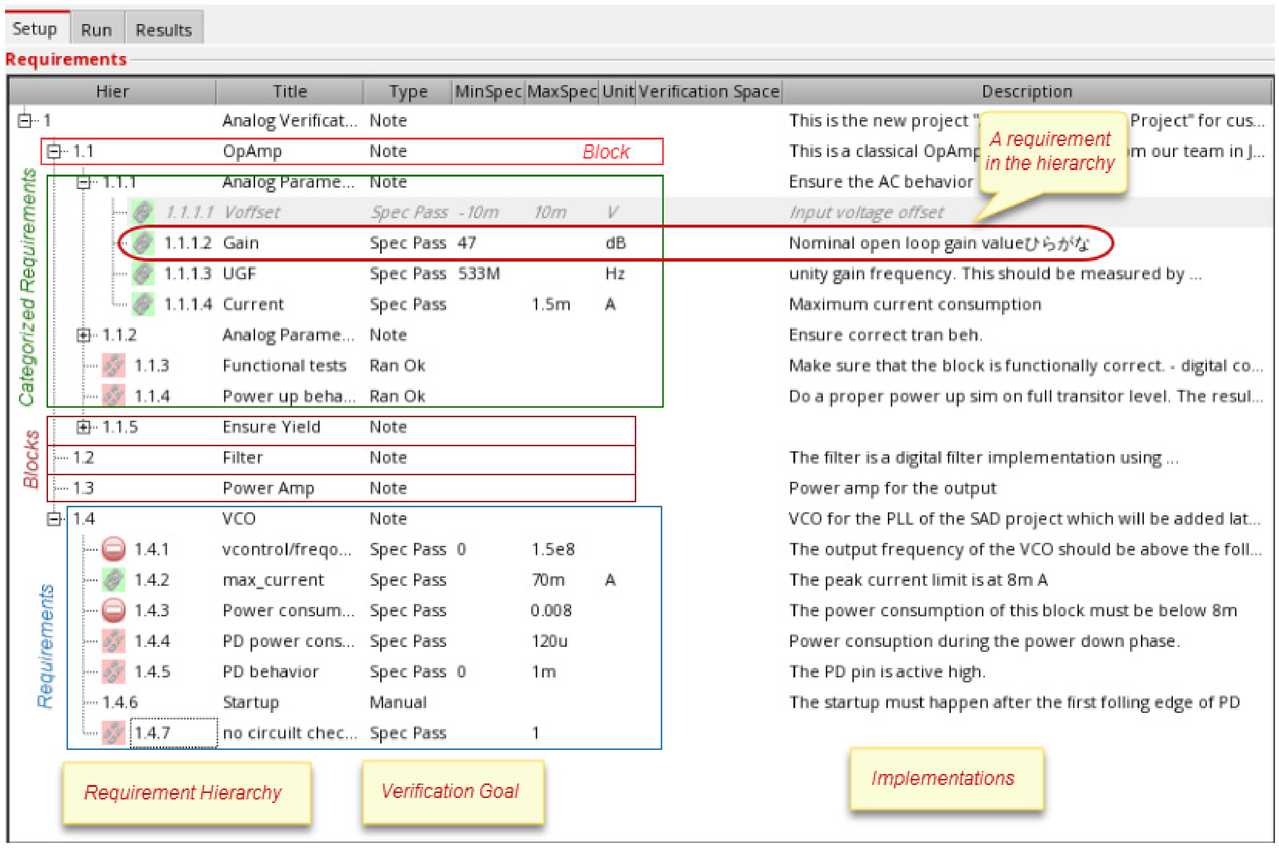
### Task 2: Review the Requirement Hierarchy

Review the hierarchical requirements tree for the design verification project.

**To review the hierarchical requirements tree**

➡ Notice that the *Setup* tab screen displays the hierarchy of design verification requirements on the left panel.

The following figure illustrates how Verifier represents the verification requirements of the sample design.



| Hier | Title | Type | MinSpec | MaxSpec | Unit | Verification Space | Description |
|---|---|---|---|---|---|---|---|
| 1 | Analog Verificat... | Note | | | | | This is the new project " Project" for cus... |
| 1.1 | OpAmp | Note | | | | | This is a classical OpAmp m our team in J... |
| 1.1.1 | Analog Parame... | Note | | | | | Ensure the AC behavior |
| 1.1.1.1 | Voffset | Spec Pass | -10m | 10m | V | | Input voltage offset |
| 1.1.1.2 | Gain | Spec Pass | 47 | | dB | | Nominal open loop gain valueひらがな |
| 1.1.1.3 | UGF | Spec Pass | 533M | | Hz | | unity gain frequency. This should be measured by ... |
| 1.1.1.4 | Current | Spec Pass | | 1.5m | A | | Maximum current consumption |
| 1.1.2 | Analog Parame... | Note | | | | | Ensure correct tran beh. |
| 1.1.3 | Functional tests | Ran Ok | | | | | Make sure that the block is functionally correct. - digital co... |
| 1.1.4 | Power up beha... | Ran Ok | | | | | Do a proper power up sim on full transitor level. The resul... |
| 1.1.5 | Ensure Yield | Note | | | | | |
| 1.2 | Filter | Note | | | | | The filter is a digital filter implementation using ... |
| 1.3 | Power Amp | Note | | | | | Power amp for the output |
| 1.4 | VCO | Note | | | | | VCO for the PLL of the SAD project which will be added lat... |
| 1.4.1 | vcontrol/freqo... | Spec Pass | 0 | 1.5e8 | | | The output frequency of the VCO should be above the foll... |
| 1.4.2 | max_current | Spec Pass | | 70m | A | | The peak current limit is at 8m A |
| 1.4.3 | Power consum... | Spec Pass | | 0.008 | | | The power consumption of this block must be below 8m |
| 1.4.4 | PD power cons... | Spec Pass | | 120u | | | Power consuption during the power down phase. |
| 1.4.5 | PD behavior | Spec Pass | 0 | 1m | | | The PD pin is active high. |
| 1.4.6 | Startup | Manual | | | | | The startup must happen after the first folling edge of PD |
| 1.4.7 | no circuit chec... | Spec Pass | | 1 | | | |

In addition to the unique auto-generated hierarchical number, unique ID, title, description, owner, and specifications, a requirement indicates the mapping status and its verification type. The following table describes the status indicators and type.

| Column | Description | |
|---|---|---|
| Mapping status icon<br><br>(displayed with the auto-generated hierarchical number) | No icon | The requirement type is set to *Note* or *Manual*. |
| | 🟩 | The requirement is mapped with one or more implementations. |
| | 🟥 | The requirement needs to be mapped, but is not mapped with any implementations. |
| | ⛔ | The specification checks failed, which indicate the following:<br>■ Verifier is set to use the preference option *Requirement specifications and check to implementation*. Verifier checks if the specifications set in the mapped requirements and implementations match. If they match, Verifier uses the specifications for validating the requirement.<br>■ The requirement meets the following conditions:<br>❑ Is mapped with one or more implementations<br>❑ Has the verification type *Spec Pass*<br>❑ The specification in the requirement does not match with the specification in the mapped implementation.<br><br>This status is the same as the implementation status *Spec Check Fail*. |
| | ↗ | The requirement is referenced from an existing Verifier cellview. |
| | ⚠ | The requirement that is referenced from an existing Verifier cellview, is out-of-date. |

| Column | Description |
|---|---|
| *Type* | Verification type of the requirement. The type can be one of the following: <br>■ *Note*: The requirement is treated as a note, without any need to check or pass any specifications. This type represents a hierarchical node in the tree, a comment, or a placeholder item. You cannot map such a requirement to an implementation. <br>■ *Spec Pass*: The requirement passes when the measured output for all the sweeps and corners that are run meet the selected specifications set in the requirements or the implementations. You can provide the specification values in the *MinSpec*, *MaxSpec*, and *Unit* fields of the requirement. Use this option when mapping a requirement with an output. <br>■ *Ran OK*: The requirement passes when the simulation is completed successfully. However, the simulation results are not checked automatically. Use this option when mapping a requirement to an implementation cellview or a test. When you map a requirement of this type to an output, Verifier checks if the measurement worked by evaluating the expression to some result, but does not check its value. <br>■ *Manual*: The requirement is marked for manual verification. You can sign off the requirement manually from the *Results* tab. You can map such a requirement with an implementation. The mapping is typically done to help find the related items that should be reviewed before signing off the requirement. <br>■ *ExtRef*: The requirement is imported in the current cellview as an external reference to an existing verifier cellview. When you import the requirement, its mapped implementations and results are also imported. You can edit the ID, Title, Specifications, and Description in the Requirement Editor for the main node of the external reference tree in the requirement hierarchy. <br>■ For details, see “Setting Verifier for External References” in the ADE Verifier User Guide. |

The requirements area includes the requirement table with details, buttons, and popup menu options to perform various operations, such as adding requirements, reorganizing the hierarchy, setting requirement owners, among other options.

**Note:** The requirements hierarchy of the blocks Filter and PowerAmp are empty because these blocks are planned in the future in the sample design project.

*Important*

The hierarchical numbers and IDs of the requirements in the sample design are the same. The hierarchical numbers are auto-generated and identify the hierarchical position of each requirement in the plan. You can edit the IDs. To view the ID, click *Show* and select *ID.*

*Tip* You can change the order of the columns shown in the displayed table by dragging the column head to the table location of your choice. This feature is available in all the tabs of Verifier. The changed order is retained only in the current session. New Verifier sessions have the default column order.

You can also hide or show the columns in the requirements and implementations tables in the *Setup* tab. For this, click *Show* and select or clear the column names. This selection is saved in the cellview.

*Tip* You can display all requirements and implementations, or display only certain categories of requirements and implementations. For this, click *Show* of the requirements or implementations area, and select the categories you want to view. The categories are:

- *Mapped*: Mapped requirements and implementations.
- *Unmapped*: Unmapped requirements and implementations.
- *Spec Check Fail*: Requirements and implementations for which the specification checks failed.

*Tip* It is possible that you have a large requirements hierarchy, implementations list, and requirement results hierarchy. In such cases, locating specific information is a difficult task. To assist you locate information, Verifier provides a filter functionality through the Requirements and Implementations panes and the toolbar.

To access this functionality, right-click anywhere on the columns headings and select *Filter Row*. This row contains individual filters for every column in the requirement hierarchy. You can enter the required criteria in a column filter to display only the requirements matching the criteria. You can also refine the filtered requirements by specifying criteria in other columns.

This row is hidden by default. See Filtering requirements based on text criteria

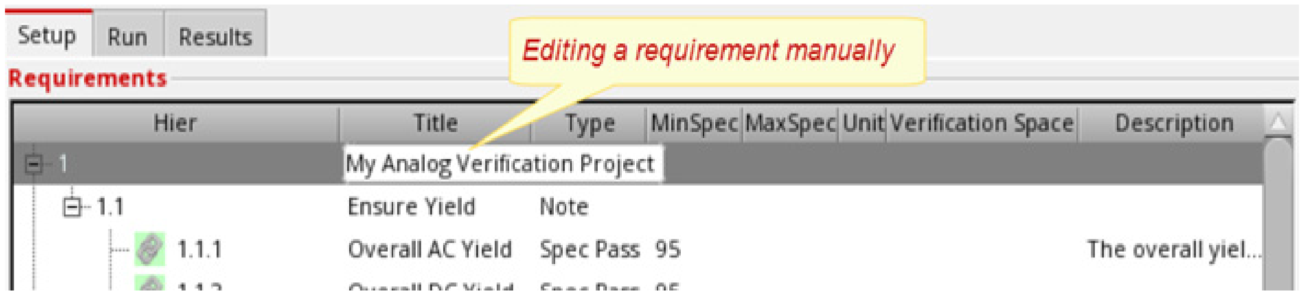
**Task 3: Edit a Requirement**

Explore how you can edit a requirement.

### To edit a requirement

1. Click the *Setup* tab.
2. Edit the requirement in the hierarchy tree directly or in the editor.

   ❑ You can edit the details in the requirements tree directly, with the exception of *Description*. Click a text field of the highlighted record and change its value. To change the goal, select the goal from the drop-down list.



| Hier | Title | Type | MinSpec | MaxSpec | Unit | Verification Space | Description |
|---|---|---|---|---|---|---|---|
| 1 | My Analog Verification Project | | | | | | |
| 1.1 | Ensure Yield | Note | | | | | |
| 1.1.1 | Overall AC Yield | Spec Pass | 95 | | | | The overall yiel... |

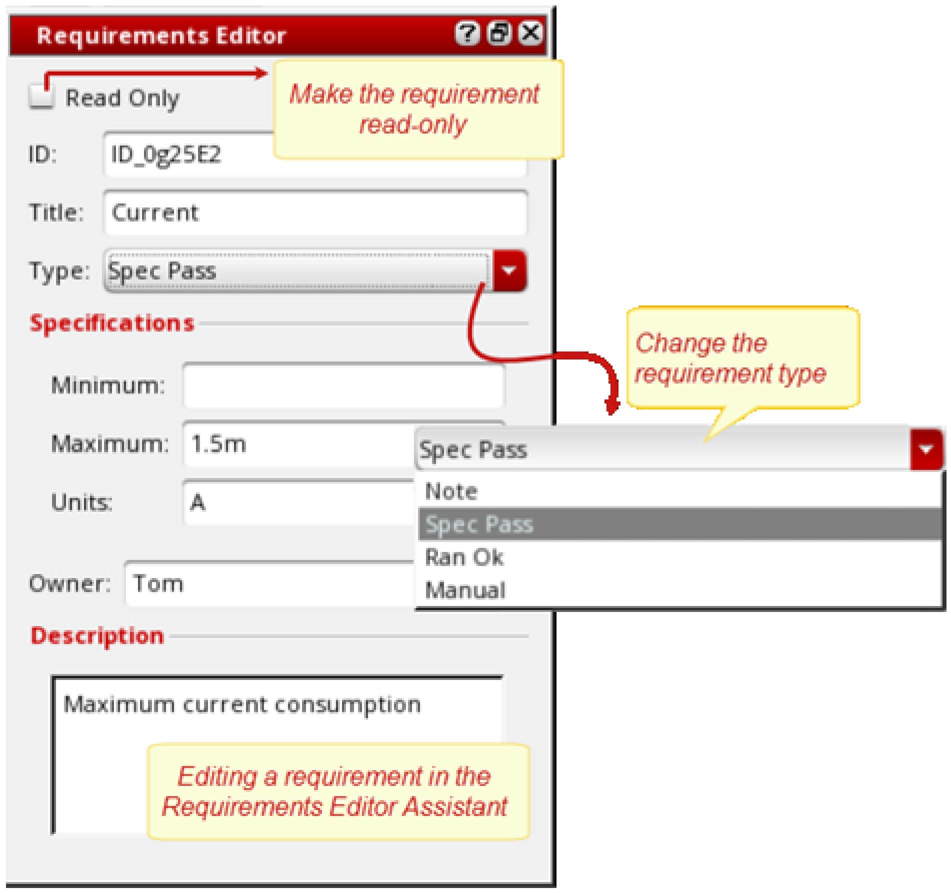
   ❑ To change the description and other details in an editor, do one of the following:

   ❍ Double-click the description field.

   ❍ Click the *Open Requirements Editor* icon on the toolbar.

   ❍ Choose *Edit — Open Requirements Editor*.

   This opens the *Requirements Editor* in the right pane where you can change the details.



**Notes:**

- You cannot edit the auto-generated hierarchical number under *Hier*.
- The editor displays in the *Setup* and *Results* tabs. It is not required in the *Run* tab.
- You can make a requirement read-only using one of the following methods:
    - Select the requirement, open the *Requirements Editor*, and enable *Read Only*.
    - Select the requirements, right-click, and select *Set Selected Requirements Read Only*.
    - Click *Show* and select *Read Only*. Select the *Read Only* check box in the requirements tree.

  Read-only requirements are distinguished in the requirements hierarchy by gray italic text and highlight. See requirement `1.1.1.1` in the sample database.

- If the custom fields feature is enabled, the editor displays the custom fields for the selected requirement.

*Tip*

If you want to retain the mapping information when renaming the requirement IDs that are mapped to their implementations, choose *Edit — Preferences*, and select the general option *Copy mapping to renamed requirement*. Save the changes.

try

Make `1.1.1.1` editable.

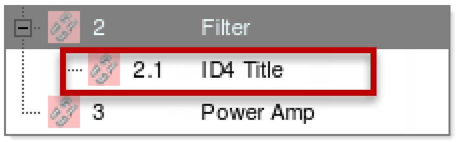
**Task 4: Add and Move a Requirement**

Add a requirement with the title `myReq` and place it between the requirements `1.2` and `1.3`.

### To add the new requirement between ID2 and ID3

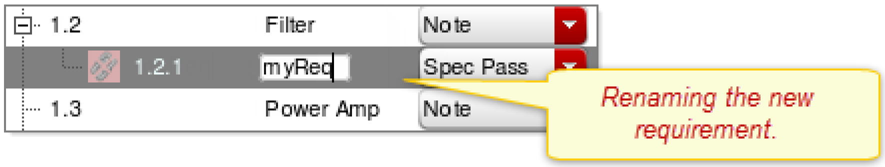
1. Do one of the following in the requirements tree of the *Setup* tab screen:
   - ❑ Right-click `1.2` and choose *Add Requirement*.
   - ❑ Select `1.2` and click *Add*.

   A new requirement appears under `1.2` called `ID5`. Note the auto-generated hierarchical number of the new requirement.



**Note:** The hierarchical number is auto-generated for all the requirements to identify their hierarchical order in the verification plan. The new requirement has the default ID format, `ID`*`Number`*, like `ID1`.

2. Select and click the new requirement. Rename the title to `myReq`.

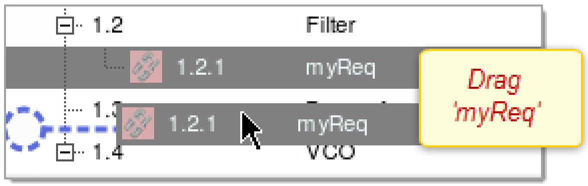


3. Do one of the following to move the new requirement at the same level as `1.2` and `1.3`. Currently, the new requirement is placed as a child requirement of `1.2`.

   ❑ Select the new requirement and click the *Decrease Indent* button.

   ❑ Select the new requirement and press `Ctrl` + `Left`.

   ❑ Drag the new requirement and place it as required.



   Notice how the hierarchical numbers of the requirements change to indicate the new hierarchical order.

**Task 5: Delete a Requirement**

Delete the requirement you added with the title `myReq` after exploring the options available to the requirement. Then save the cellview.

### To delete a requirement in the *Setup* tab screen

1. Right-click the requirement (`myReq` for the task) and choose *Delete Requirements*.

   **Note:** You can select multiple requirements for deletion.

2. Click *Yes*, when prompted, to confirm the deletion.

You can click *File — Save* to save the cellview.

The hierarchical numbers of the requirements change to indicate the new hierarchical order.

## Working with Implementations

This section details the following tasks.

- Task 6: Review the Implementations List
- Task 7: Review the Requirement–Implementation Mappings
- Task 8: Add an Implementation Manually
- Task 9: Define Requirement–Implementation Mappings
- Task 10: Check and Implement Verifier Specifications

### Task 6: Review the Implementations List

In the sample design, consider that designers have already accomplished certain tasks so that the related implementation data is already specified and available. The cellview `amsPLL/TOP_verification/verification` contains such implementations.

Open the cellview, if it is not already displayed, and explore the implementations.

#### To explore the implementation data in the cellview

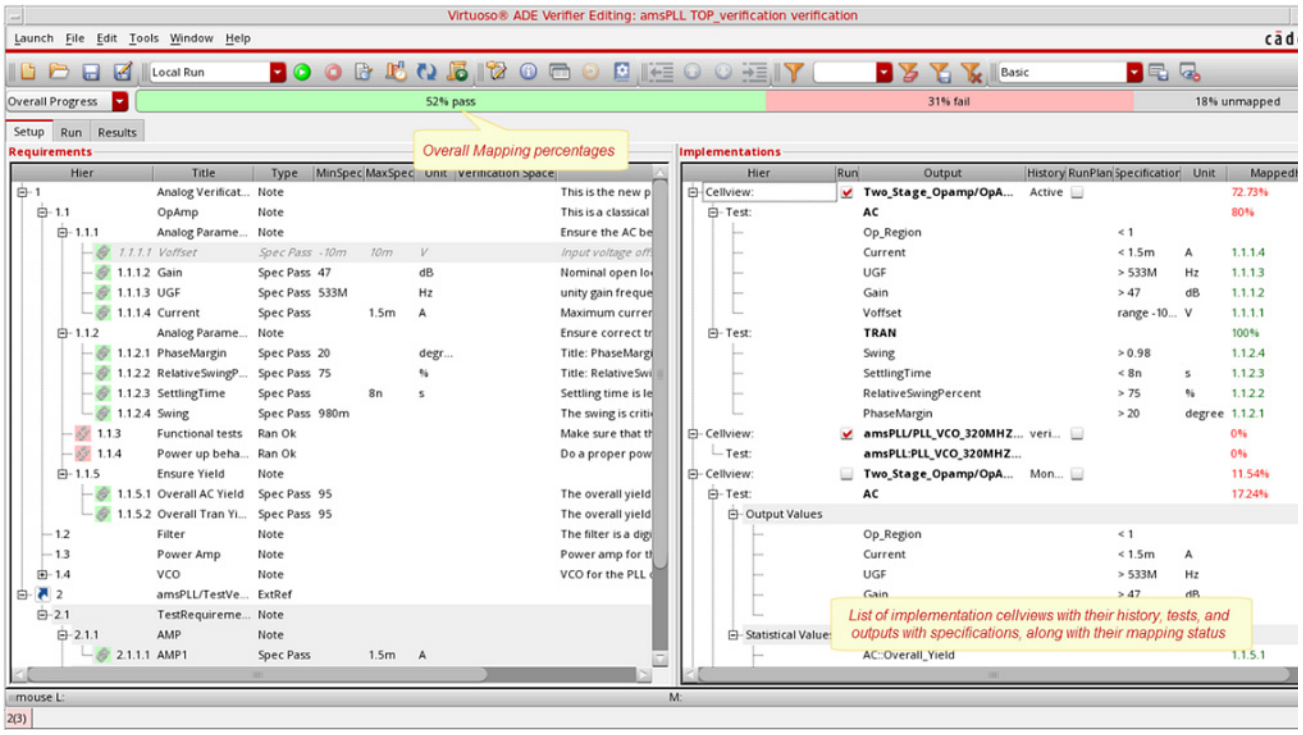
1. Click the *Setup* tab.

   If the *Requirement Editor* is displayed, hide it by clicking *Hide* in the editor.

   The right pane displays the implementations that you can map with the requirements. The left pane displays the requirements hierarchy.

   The *MappedHier* column in the implementations pane displays the individual mapping percentages. The *Overall Progress* bar displays the overall mapping percentages.

The following figure illustrates the implementations of the sample design.


Virtuoso® ADE Verifier Editing: amsPLL TOP_verification verification
Launch File Edit Tools Window Help
Local Run
Basic
Overall Progress
52% pass
31% fail
18% unmapped
Overall Mapping percentages
Setup Run Results
Requirements
Implementations
List of implementation cellviews with their history, tests, and outputs with specifications, along with their mapping status
mouse L:
M:
2(3)

| Hier | Title | Type | MinSpec | MaxSpec | Unit | Verification Space | |
|---|---|---|---|---|---|---|---|
| 1 | Analog Verificat... | Note | | | | | This is the new p |
| 1.1 | OpAmp | Note | | | | | This is a classical |
| 1.1.1 | Analog Parame... | Note | | | | | Ensure the AC be |
| 1.1.1.1 | *Voffset* | *Spec Pass* | *-10m* | *10m* | *V* | | *Input voltage off* |
| 1.1.1.2 | Gain | Spec Pass | 47 | | dB | | Nominal open lo |
| 1.1.1.3 | UGF | Spec Pass | 533M | | Hz | | unity gain freque |
| 1.1.1.4 | Current | Spec Pass | | 1.5m | A | | Maximum currer |
| 1.1.2 | Analog Parame... | Note | | | | | Ensure correct tr |
| 1.1.2.1 | PhaseMargin | Spec Pass | 20 | | degr... | | Title: PhaseMargi |
| 1.1.2.2 | RelativeSwingP... | Spec Pass | 75 | | % | | Title: RelativeSwi |
| 1.1.2.3 | SettlingTime | Spec Pass | | 8n | s | | Settling time is le |
| 1.1.2.4 | Swing | Spec Pass | 980m | | | | The swing is criti |
| 1.1.3 | Functional tests | Ran Ok | | | | | Make sure that th |
| 1.1.4 | Power up beha... | Ran Ok | | | | | Do a proper pow |
| 1.1.5 | Ensure Yield | Note | | | | | |
| 1.1.5.1 | Overall AC Yield | Spec Pass | 95 | | | | The overall yield |
| 1.1.5.2 | Overall Tran Yi... | Spec Pass | 95 | | | | The overall yield |
| 1.2 | Filter | Note | | | | | The filter is a dig |
| 1.3 | Power Amp | Note | | | | | Power amp for th |
| 1.4 | VCO | Note | | | | | VCO for the PLL |
| 2 | amsPLL/TestVe... | ExtRef | | | | | |
| 2.1 | TestRequireme... | Note | | | | | |
| 2.1.1 | AMP | Note | | | | | |
| 2.1.1.1 | AMP1 | Spec Pass | | 1.5m | A | | |

| Hier | Run | Output | History | RunPlan | Specification | Unit | Mapped |
|---|---|---|---|---|---|---|---|
| Cellview: | ✓ | **Two_Stage_Opamp/OpA...** | Active | ☐ | | | 72.73% |
| Test: | | **AC** | | | | | 80% |
| | | Op_Region | | | < 1 | | |
| | | Current | | | < 1.5m | A | 1.1.1.4 |
| | | UGF | | | > 533M | Hz | 1.1.1.3 |
| | | Gain | | | > 47 | dB | 1.1.1.2 |
| | | Voffset | | | range -10... | V | 1.1.1.1 |
| Test: | | **TRAN** | | | | | 100% |
| | | Swing | | | > 0.98 | | 1.1.2.4 |
| | | SettlingTime | | | < 8n | s | 1.1.2.3 |
| | | RelativeSwingPercent | | | > 75 | % | 1.1.2.2 |
| | | PhaseMargin | | | > 20 | degree | 1.1.2.1 |
| Cellview: | ✓ | **amsPLL/PLL_VCO_320MHZ...** | veri... | ☐ | | | 0% |
| Test: | | **amsPLL:PLL_VCO_320MHZ...** | | | | | 0% |
| Cellview: | ☐ | **Two_Stage_Opamp/OpA...** | Mon... | ☐ | | | 11.54% |
| Test: | | **AC** | | | | | 17.24% |
| Output Values | | | | | | | |
| | | Op_Region | | | < 1 | | |
| | | Current | | | < 1.5m | A | |
| | | UGF | | | > 533M | Hz | |
| | | Gain | | | > 47 | dB | |
| Statistical Value | | | | | | | |
| | | AC::Overall_Yield | | | | | 1.1.5.1 |

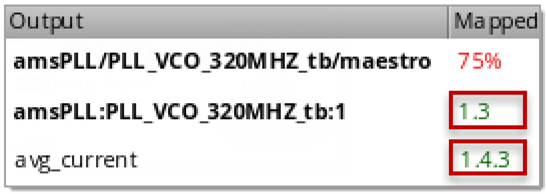
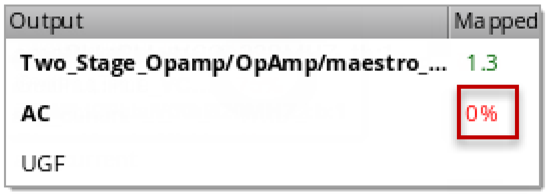
**2.** Review the implementations list.

**Note:** You can show or hide the columns. For details, see the related tip.

The implementation hierarchy contains the following details.

| Column | Description |
|---|---|
| *Hier* | The hierarchy of the implementation cellview, in relation with its tests and outputs. |
| *Library, Cell, View* | The library, cell, and view of the implementation cellview. These columns are hidden by default. |

| Column | Description |
|---|---|
| *History* | The option to select the cellview history.<br><br>When you initiate the simulation of an implementation from Verifier, a session is loaded with the implementation cellview where the specified history is set as the active history within that session. By default, a new history is created outside the cellview as the implementation is run.<br><br>**Notes:**<br>■ The cellview–history entry must be unique in the implementations list.<br>■ Verifier displays the tests and outputs of the selected history, which you can map to requirements. |
| *Run* | The option to run the implementation from Verifier. Select this check box if you want to initiate the run from Verifier, or clear it to load the test results from the selected history when the implementation cellview is run from outside Verifier. |
| *Test* | The tests of the implementation cellview. This column is hidden by default. |
| *Output* | The name of the simulation output to be used for determining the verification result.<br><br>Outputs are measurements and assertions that can be evaluated using the simulation results. If the run mode of the implementation cellview is *Monte Carlo Sampling*, the outputs include the output values and statistical values. |
| *Specification* | The performance specification to determine if the related simulation result passed.<br><br>This column displays *Spec Check Fail* to indicate issues found in its specification checks. This is similar to the requirement status that indicates specification check failure. |
| *Units* | The unit used for the measurement output. |

| Column | Description |
|---|---|
| *MappedHier* | The mapping status of the implementation.<br><br>**Note:** The requirements hierarchy indicates the mapping status through icons. The title of the requirements area and implementations area indicate the overall mapping percentage.<br><br>The following are the possible mapping status of implementations:<br><br>■ *Requirement ID*: Indicates that the implementation is mapped to the requirement with the specified ID.<br><br><br><br>Output \| Mapped<br>amsPLL/PLL_VCO_320MHZ_tb/maestro \| 75%<br>amsPLL:PLL_VCO_320MHZ_tb:1 \| 1.3<br>avg_current \| 1.4.3<br><br>■ *Percentage*: Indicates the percentage of the mapped implementations under the cellview or test. The percentage is displayed as a note when a cellview or a test is not mapped.<br><br><br><br>Output \| Mapped<br>Two_Stage_Opamp/OpAmp/maestro_... \| 1.3<br>AC \| 0%<br>UGF<br><br>**Note:** It is not necessary to achieve 100% mapping for all implementations or requirements. You can aim to achieve 100% mapping if you want to ensure that each test and each output in each implementation cellview is mapped to a requirement. The status of your verification project depends largely on the requirements and their mappings. |

The implementation area provides buttons and popup menu options to perform various operations, such as adding or deleting implementations, overriding specifications, defining requirement–specification mappings, and opening implementation cellviews.

*Tip* Verifier lets you search the implementations list, which is similar to the search functionality for requirements and results.

Verifier also lets you hide or show the implementation tests and outputs that are not mapped to requirements. To hide unmapped implementation tests and outputs, right-click, and select *Show* from and select *Unmapped* from the context menu to remove the check. Hidden tests and outputs are not considered when calculating the percentage of mapped implementations.

To show hidden and unmapped implementation tests and outputs, right-click the implementations area, select *Show — Spec Check Fail* the to show implementations that have failed the specifications verification. To hide implementations, clear their check marks.

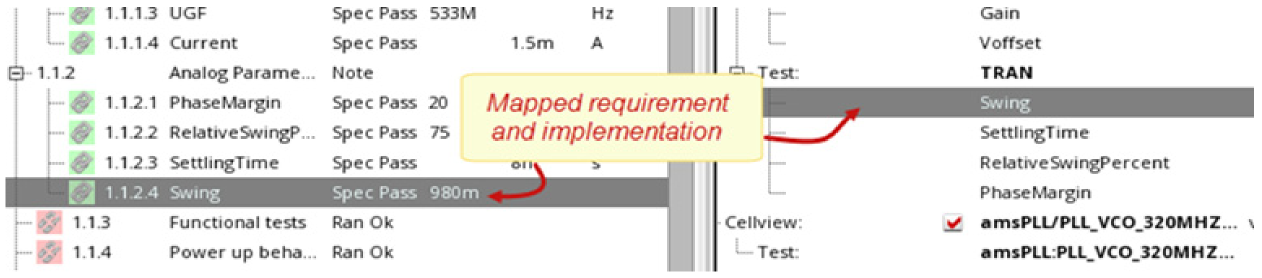
**Task 7: Review the Requirement–Implementation Mappings**

Review the existing requirement–implementation mappings in the cellview `amsPLL/TOP_verification/verification`. Identify the implementation that is mapped to requirement `1.1.1.2`.

### To review requirement–implementation mappings

1. Select *Show Cross Mapping at Cursor* in the *Edit — Preferences — General* tab screen.
2. Place the cursor over the requirement or implementation to identify their mapping, as illustrated in the following figure.

For completing this task, place the cursor over `1.1.1.2`. It is mapped to this implementation: `Two_Stage_OpAmp/maestro_nominal/Active.AC.Gain`



| | | | | | |
|---|---|---|---|---|---|
| 1.1.1.3 | UGF | Spec Pass | 533M | | Hz |
| 1.1.1.4 | Current | Spec Pass | | 1.5m | A |
| 1.1.2 | Analog Parame... | Note | | | |
| 1.1.2.1 | PhaseMargin | Spec Pass | 20 | | |
| 1.1.2.2 | RelativeSwingP... | Spec Pass | 75 | | |
| 1.1.2.3 | SettlingTime | Spec Pass | | 6n | s |
| 1.1.2.4 | Swing | Spec Pass | 980m | | |
| 1.1.3 | Functional tests | Ran Ok | | | |
| 1.1.4 | Power up beha... | Ran Ok | | | |

| | |
|---|---|
| | Gain |
| | Voffset |
| Test: | TRAN |
| | Swing |
| | SettlingTime |
| | RelativeSwingPercent |
| | PhaseMargin |
| Cellview: | ✓ amsPLL/PLL_VCO_320MHZ... |
| Test: | amsPLL:PLL_VCO_320MHZ... |

When you click a requirement or implementation, the mapped entries are selected. You can identify other mappings by placing the cursor over other requirements or implementations.

*Tip*

You can choose your preference to use implementation specifications, or the specifications defined in requirements for determining the verification status. For this, choose *Edit — Preferences* to open the form Virtuoso ADE Verifier Preferences, and set your preference in *General Options — Specification Search Order*. The following are the available options:

- *Requirement specification and check to implementation*: Verifier checks if the specifications set in the requirements match with the specifications of the mapped implementations. Verifier also uses the specified values to determine the verification results.

- *Requirement specification*: Verifier uses the minimum and maximum values set in the requirements to determine the verification results. If a requirement does not have any specification, Verifier uses the specification set in the mapped implementation to determine the verification result of that requirement.

  **Note:** The specification units are not used in the specification comparison. They are used for documentation purposes and when the requirement and implementation specifications are compared.

- *Implementation specifications*: Verifier uses the specifications set in the mapped implementations to determine the requirement verification results. If an implementation does not have any specification values, Verifier uses the values set in the requirement mapped to that implementation.

- *Check units:* Verifier uses this option to check the units specified in the requirements and implementations.

**Task 8: Add an Implementation Manually**

Add `amsPLL/PLL_VCO_320MHZ_PD_tb/maestro` as an implementation cellview in the implementations list.

### To add an implementation cellview

1. Click *Add Implementation* in the implementations pane.

   The *Add Implementation* form displays.

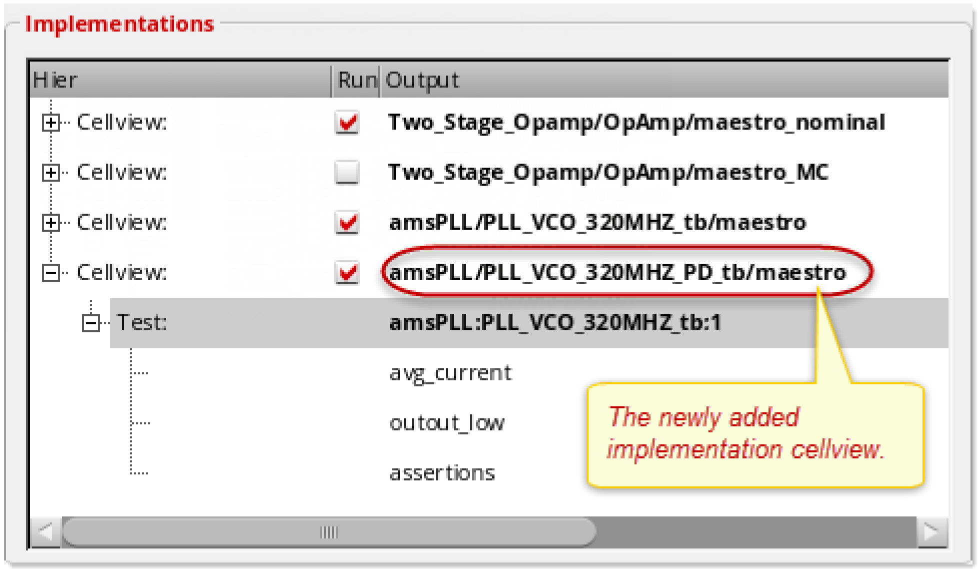
2. Specify the library, cell, view and history to select the implementation cellview.

   For the task, select `amsPLL/PLL_VCO_320MHZ_PD_tb/maestro`.

3. Click *OK*.

   If the selected cellview has a single `Active` history, it is used as the default history for the implementation. If it has multiple histories, the Implementation History Item form displays. Select the history that you want to use and click *OK*.

The implementation cellview appears as the last entry in the implementations list.



| Hier | Run | Output |
|---|---|---|
| Cellview: | ✓ | Two_Stage_Opamp/OpAmp/maestro_nominal |
| Cellview: | | Two_Stage_Opamp/OpAmp/maestro_MC |
| Cellview: | ✓ | amsPLL/PLL_VCO_320MHZ_tb/maestro |
| Cellview: | ✓ | amsPLL/PLL_VCO_320MHZ_PD_tb/maestro |
| Test: | | amsPLL:PLL_VCO_320MHZ_tb:1 |
| | | avg_current |
| | | outout_low |
| | | assertions |

*Tip*

To change the history of an implementation cellview with multiple histories, display the *History* column if it is hidden and select the history you want to use.

### Task 9: Define Requirement–Implementation Mappings

The new implementation cellview `amsPLL/PLL_VCO_320MHZ_PD_tb/maestro` has three outputs. Map these outputs to the requirements mentioned below:

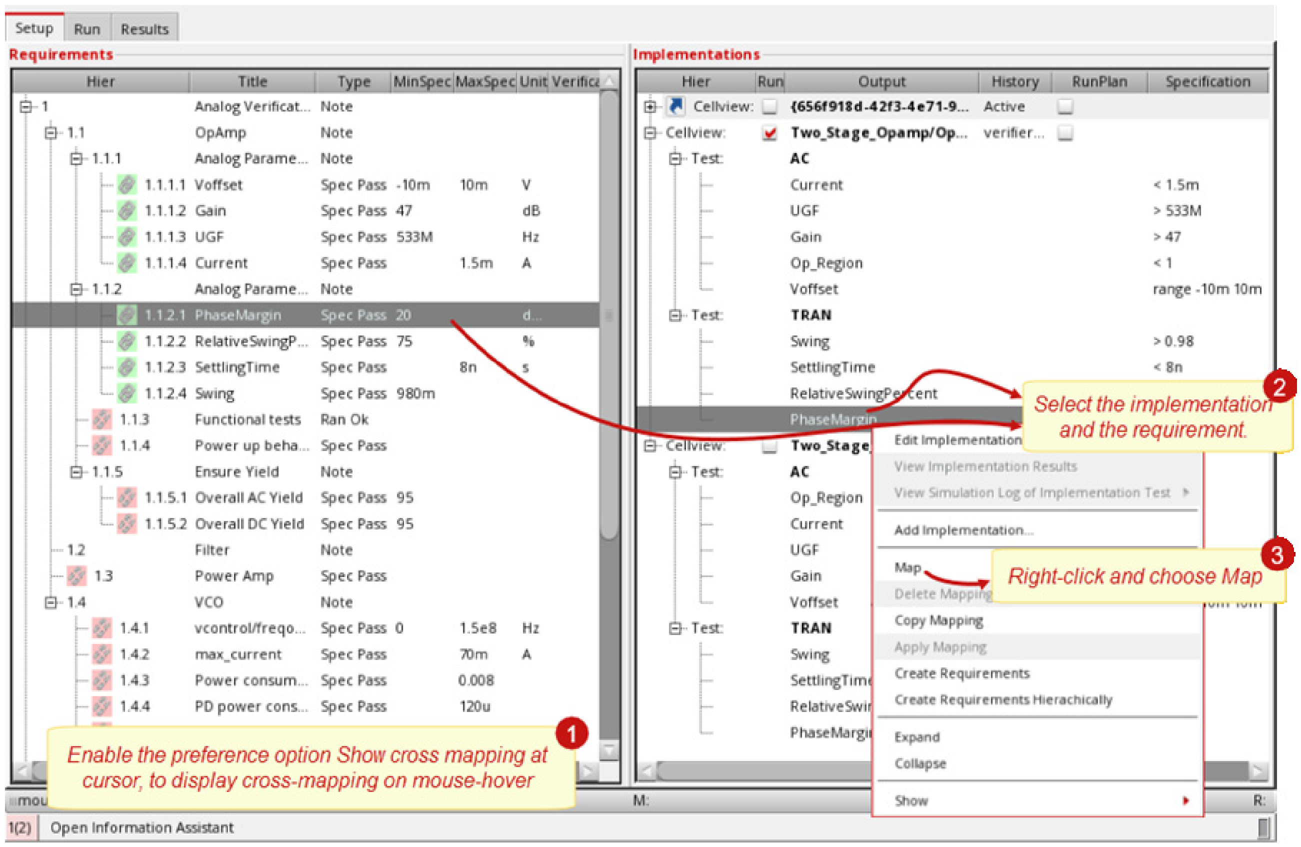
- Map `avg_current` to the requirement `1.4.4`.
- Map `outout_low` to the requirement `1.4.5`.
- Map `assertions` to the requirement with the ID `1.4.7`.

## To map a requirement to an implementation

1. Select the requirement and the implementation in the *Setup* tab.

   **Note:** Hold down the `Ctrl` key and select the requirement and implementation. You can select multiple records, unless the preference option *Map only one requirement to one implementation* is enabled.

2. Click *Map*.


Setup
Run
Results
Requirements
Implementations
Enable the preference option Show cross mapping at cursor, to display cross-mapping on mouse-hover
Select the implementation and the requirement.
Right-click and choose Map
Edit Implementation
View Implementation Results
View Simulation Log of Implementation Test
Add Implementation...
Map
Delete Mapping
Copy Mapping
Apply Mapping
Create Requirements
Create Requirements Hierachically
Expand
Collapse
Show
Open Information Assistant

| Hier | Title | Type | MinSpec | MaxSpec | Unit |
|---|---|---|---|---|---|
| 1 | Analog Verificat... | Note | | | |
| 1.1 | OpAmp | Note | | | |
| 1.1.1 | Analog Parame... | Note | | | |
| 1.1.1.1 | Voffset | Spec Pass | -10m | 10m | V |
| 1.1.1.2 | Gain | Spec Pass | 47 | | dB |
| 1.1.1.3 | UGF | Spec Pass | 533M | | Hz |
| 1.1.1.4 | Current | Spec Pass | | 1.5m | A |
| 1.1.2 | Analog Parame... | Note | | | |
| 1.1.2.1 | PhaseMargin | Spec Pass | 20 | | d... |
| 1.1.2.2 | RelativeSwingP... | Spec Pass | 75 | | % |
| 1.1.2.3 | SettlingTime | Spec Pass | | 8n | s |
| 1.1.2.4 | Swing | Spec Pass | 980m | | |
| 1.1.3 | Functional tests | Ran Ok | | | |
| 1.1.4 | Power up beha... | Spec Pass | | | |
| 1.1.5 | Ensure Yield | Note | | | |
| 1.1.5.1 | Overall AC Yield | Spec Pass | 95 | | |
| 1.1.5.2 | Overall DC Yield | Spec Pass | 95 | | |
| 1.2 | Filter | Note | | | |
| 1.3 | Power Amp | Spec Pass | | | |
| 1.4 | VCO | Note | | | |
| 1.4.1 | vcontrol/freqo... | Spec Pass | 0 | 1.5e8 | Hz |
| 1.4.2 | max_current | Spec Pass | | 70m | A |
| 1.4.3 | Power consum... | Spec Pass | | 0.008 | |
| 1.4.4 | PD power cons... | Spec Pass | | 120u | |

| Hier | Run | Output | History | Specification |
|---|---|---|---|---|
| Cellview: | | {656f918d-42f3-4e71-9... | Active | |
| Cellview: | ✓ | Two_Stage_Opamp/Op... | verifier... | |
| Test: | | AC | | |
| | | Current | | < 1.5m |
| | | UGF | | > 533M |
| | | Gain | | > 47 |
| | | Op_Region | | < 1 |
| | | Voffset | | range -10m 10m |
| Test: | | TRAN | | |
| | | Swing | | > 0.98 |
| | | SettlingTime | | < 8n |
| | | RelativeSwingPercent | | |
| | | PhaseMargin | | |
| Cellview: | | Two_Stage | | |
| Test: | | AC | | |
| | | Op_Region | | |
| | | Current | | |
| | | UGF | | |
| | | Gain | | |
| | | Voffset | | |
| Test: | | TRAN | | |
| | | Swing | | |
| | | SettlingTim | | |
| | | RelativeSwi | | |
| | | PhaseMargi | | |

*Important*

The ID of a requirement is used for the mapping definition. The hierarchical number is not used because it can change based on modifications in the verification plan.

try

To remove requirement–implementation mappings, select the requirements or implementations, right-click, and choose *Delete Mapping*.

### Task 10: Check and Implement Verifier Specifications

Choose to check and use the specifications set in the requirements by selecting *General Options — Requirement specifications and check to implementation* in the Virtuoso ADE Verifier Preferences form. Also, enable the option *Check units* to check if the units mentioned in the requirements and implementations match. For details, see the related tip on preferences.

The specification check for the requirement `1.4.1` has failed. This is indicated by the requirement status icon and the value in *Specification* field of the mapped implementation. Identify the reason of this failure by reviewing the logs and other settings. Then, rectify the settings so that the specification check for `1.4.1` passes.

**To view the log and identify the reason of the specification check failure**

1. Choose *View — Show Log*.
2. Review the log entries to identify the issue.

   The following example log entry illustrates how you can identify the issue for `1.4.5`. It indicates the reason for the failure of the specification check for the `1.4.5-PD behaviour` mapping.

```
13:47:05: ERROR (Verifier-80146): Unit '' of requirement '1.4.1' does not
match with unit 'Hz' of the mapped implementation 'amsPLL/PLL_VCO_320MHZ_tb/
maestro.amsPLL:PLL_VCO_320MHZ_tb:1.freq_check'.

Mismatch information

Verification requirement '1.4.1'
amsPLL/PLL_VCO_320MHZ_tb/maestro.amsPLL:PLL_VCO_320MHZ_tb:1.freq_check
        Title                Requirement Spec          Implementation Spec
        vcontrol/freqout beh  0 to 1.5e8               0 to 150M (Hz)
```

**Notes:**

- ❑ In the Virtuoso ADE Verifier Preferences form, if you disable the *Check units* option under the *General Options* tab, this specification check failure due to mismatching units will not occur.
- ❑ Place the cursor over a requirement to see the details, including the reason of the specification check failure.

---

try

Choose *Edit — Preferences*, click *Implementation Specifications* in the *General Options* tab, and click *OK*. The specification check failure indicator of `1.4.1` is removed.

Hide the log area. Then, reset the Verifier preference to use the specifications set in requirements and check the units. The specification check failure icon reappears for `1.4.1`. Also, the log and save buttons on the toolbar get highlighted with a red background. The red log button indicates that a log entry requires your attention. In this case, that log entry is about the specification check failure. Similarly, the red save icon indicates that the cellview contains unsaved changes.

---

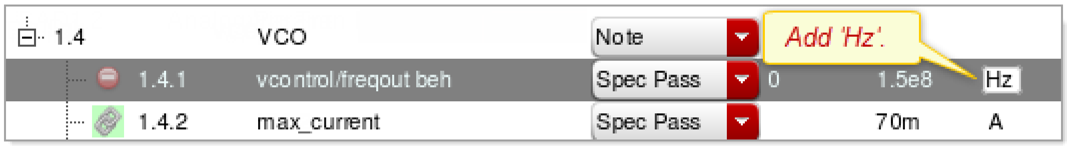
### To investigate and correct the specification check failure

1. Review the reason of the failure using the log.
2. Decide the approach you want to use to rectify the failure and use that approach.

   For example, to resolve the issue of the `1.4.1-freq_check` mapping, you can take one of the following actions:

   - ❑ Change the requirement so that its specification matches with the specification of the corresponding `freq_check` output.

     For this task, add the unit `Hz` to the requirement.



| | | | | | |
|---|---|---|---|---|---|
| 1.4 | VCO | Note | | | |
| 1.4.1 | vcontrol/freqout beh | Spec Pass | 0 | 1.5e8 | Hz |
| 1.4.2 | max_current | Spec Pass | | 70m | A |

The icon changes. The *Overall Progress* also changes from 52 percent to 57 percent.

❑ Change the setup of output `amsPLL/PLL_VCO_320MHZ_tb/maestro/Active.amsPLL:PLL_VCO_320MHZ_tb:1.freq_check` so that it matches with the requirement specification.

Verifier checks for changes in implementation cellviews frequently and updates information available to the Verifier session accordingly. You can also choose *Tools — Check for Changes in Implementations* in Verifier.

*Tip*

You can open an implementation cellview to change the output setup. For this, right-click the cellview and choose *Edit Implementation Cellview*. Make the changes and save the cellview.

*Tip*

If the preference option *Time Interval Between Implementation Change Checks* is set to `0` in the Virtuoso ADE Verifier Preferences form, update the information available to Verifier manually.

*Tip*

You can overwrite the specification set in an implementation with the specification set in its mapped requirement. Likewise, you can overwrite the specification set in a requirement with the specification set in its mapped implementation. For overwriting the specification, select the relevant implementations and right-click to display the popup menu. Then, select *Overwrite Implementation Specification* or *Overwrite Verifier Specifications* based on your overwrite requirement.

## Simulating Implementation Cellviews

This section details the following tasks.

- Task 11: Ensure that Verifier Can Initiate Simulations
- Task 12: Organize Implementation Cellviews in Implementation Sets
- Task 13: Simulate Implementation Cellviews
- Task 14: Monitor Simulation Status
- Task 15: View Simulation Details

**Note:** For information on using a batch script to run simulations, see "Generating and Using Batch Scripts".

### Task 11: Ensure that Verifier Can Initiate Simulations

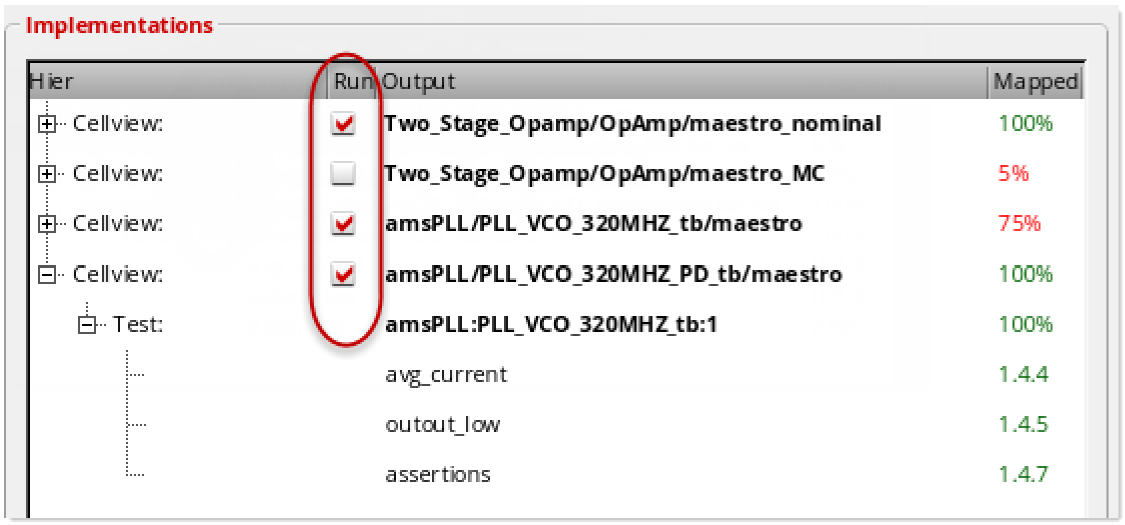
The cellview `amsPLL/TOP_verification/verification` of the sample design database now includes four implementation cellviews with tests. Ensure that you can start the simulation of the following three implementation cellviews from Verifier:

- `Two_Stage_Opamp/OpAmp/maestro_nominal/Active`
- `amsPLL/PLL_VCO_320MHZ_tb/maestro/Active`
- `amsPLL/PLL_VCO_320MHZ_PD_tb/maestro/Active` (added in Task 8)

For `Two_Stage_Opamp/OpAmp/maestro_MC/Active`, choose to simulate the implementation cellview from an external application and load the simulation results in Verifier.

### To ensure that you can start the simulations of implementation cellviews from Verifier

1. Click the *Setup* tab to view the implementations.
2. Ensure that the *Run* check box is selected for all the implementation cellviews that you want to run from Verifier.


Implementations

| Hier | Run | Output | Mapped |
|---|---|---|---|
| Cellview: | ✓ | Two_Stage_Opamp/OpAmp/maestro_nominal | 100% |
| Cellview: | ☐ | Two_Stage_Opamp/OpAmp/maestro_MC | 5% |
| Cellview: | ✓ | amsPLL/PLL_VCO_320MHZ_tb/maestro | 75% |
| Cellview: | ✓ | amsPLL/PLL_VCO_320MHZ_PD_tb/maestro | 100% |
| Test: | | amsPLL:PLL_VCO_320MHZ_tb:1 | 100% |
| | | avg_current | 1.4.4 |
| | | outout_low | 1.4.5 |
| | | assertions | 1.4.7 |

**Note:** For the implementation cellview `Two_Stage_Opamp/OpAmp/maestro_MC`, clear the *Run* check box to inform Verifier that this cellview is to be simulated from an external application. After the simulation is completed outside Verifier, you can load the results stored in a cellview history. If required, you can select the history from the *History* options of the implementation. To reload the results, click the *Reload Simulation Results* button on the toolbar, or choose *Tools — Reload Implementation Simulation Results*. You can also load results from the *Run* tab.

**Task 12: Organize Implementation Cellviews in Implementation Sets**

The implementation cellviews in the *Run* tab are organized in the *Implementation Sets*, namely `run_weekly` and `run_daily`. Also, all the four implementation cellviews of the verification project are listed in the *Implementations* tree.

In this task, try adding and removing an implementation set called `mySet`. Then, add the implementation cellview `amsPLL/PLL_VCO_320MHZ_PD_tb/maestro` to the implementation set *run_weekly*.

### To add an implementation set

1. Click the *Run* tab.
2. Right-click in the left pane and choose *Create New Implementation Set...* from the context menu.

   The Implementation Set Name form displays.
3. Type the name of the implementation set.

   For this task, type `mySet`.
4. Click *OK*.

The implementation set is added. If some implementation cellview in the *Implementations* tree were selected, they get included in the new implementation set. If you had selected another implementation set, then the new set is added as a child set of the selected set.

### To delete an implementation set

1. Right-click the implementation set and choose *Delete from Implementation Set...* from the context menu.
2. When prompted, click *Yes* to confirm the deletion.

   For this task, remove `mySet`.

The implementation set and its implementation cellviews are removed. The implementation cellviews continue to be available under the *Implementations* tree.

**Notes:**

- An implementation cellview can belong to multiple implementation sets.
- You can rearrange an implementation set or cellview in the *Implementation Sets* tree using the up or down arrow buttons. You cannot rearrange the cellviews under *Implementations*. However, when you rearrange the implementation cellviews from the *Setup* tab.

### To add an implementation cellview to an implementation set

1. Right-click the cellview in the *Implementations* tree of the *Run* tab and choose *Add to Implementation Set*. The available implementation sets are displayed in the context menu.

   For this task, right-click `amsPLL/PLL_VCO_320MHZ_PD_tb/maestro` and then choose that option.

   **Note:** You can select multiple implementation cellviews, right-click, and choose *Add to Implementation Set*.

2. Select the implementation set from the context menu.

   For this task, select *run_weekly*.

3. Click *OK*.

The selected implementation cellview is organized in the implementation set.

### Task 13: Simulate Implementation Cellviews

Verifier lets you run a single implementation cellview, all cellviews, or cellviews that belong to an implementation set.

You can load the simulation results of `Two_Stage_Opamp/OpAmp/maestro_MC` by clicking its button to load the results in the *Run* tab. You can simulate the other implementation cellviews from Verifier.

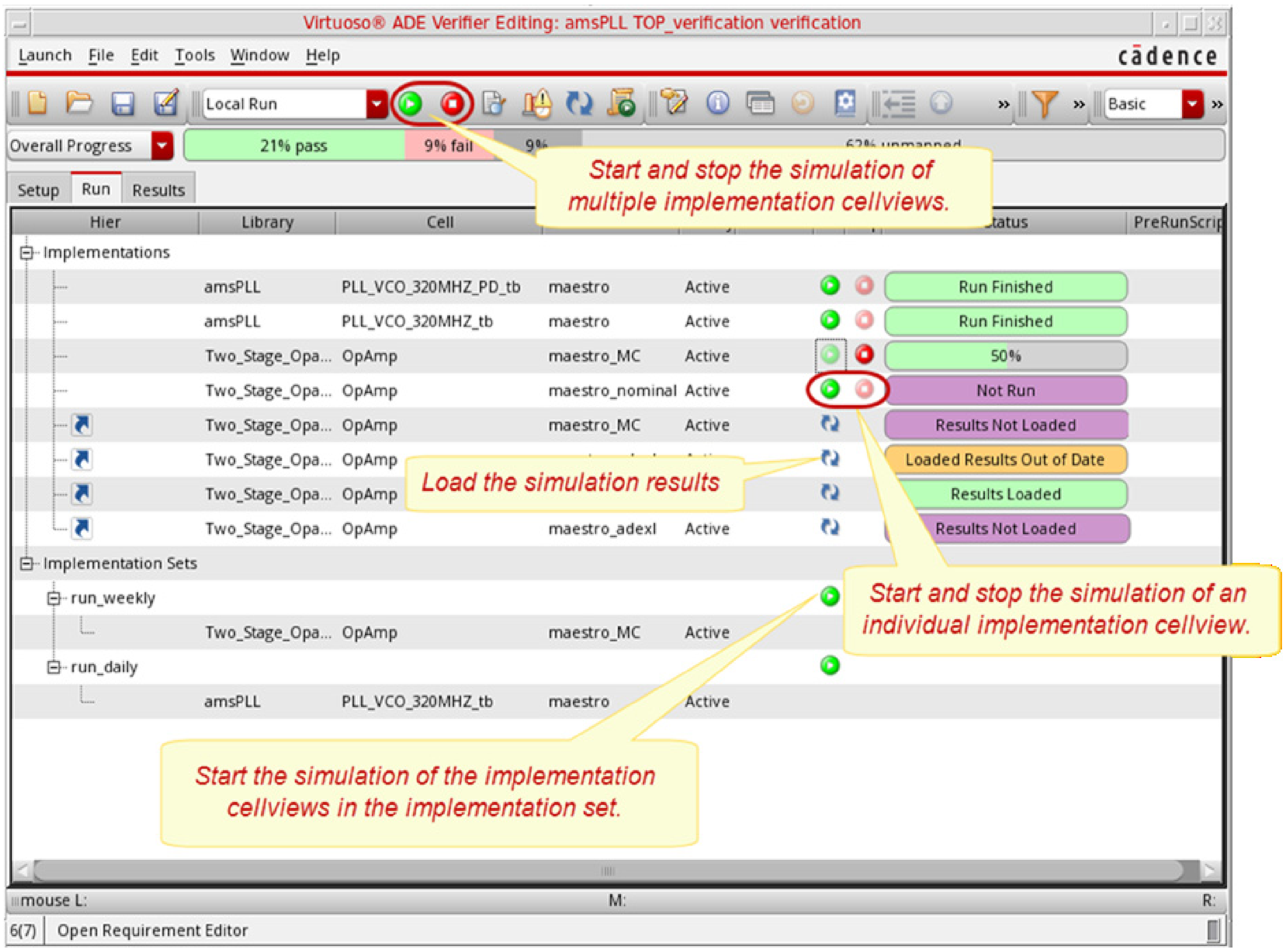
In this task, run all the cellviews that belong to the implementation set *run_weekly*.

### To start simulating the implementation cellviews from Verifier

1. Click the *Run* tab.

   You can start the simulation of multiple implementation cellviews from Verifier serially or in parallel. You can also load the simulation results stored in the history of an implementation cellview. You typically organize implementation cellviews in implementation sets and then choose to run all the implementation cellviews of an implementation set, as required.

The following image displays the *Run* tab.



Virtuoso® ADE Verifier Editing: amsPLL TOP_verification verification

Launch File Edit Tools Window Help — cādence

Local Run — Basic

Overall Progress: 21% pass | 9% fail | 9% | 62% unmapped

Setup | Run | Results

Start and stop the simulation of multiple implementation cellviews.

| Hier | Library | Cell | | | | | tatus | PreRunScrip |
|---|---|---|---|---|---|---|---|---|
| Implementations | | | | | | | | |
| | amsPLL | PLL_VCO_320MHZ_PD_tb | maestro | Active | ● | ● | Run Finished | |
| | amsPLL | PLL_VCO_320MHZ_tb | maestro | Active | ● | ● | Run Finished | |
| | Two_Stage_Opa... | OpAmp | maestro_MC | Active | ● | ● | 50% | |
| | Two_Stage_Opa... | OpAmp | maestro_nominal | Active | ● | ● | Not Run | |
| | Two_Stage_Opa... | OpAmp | maestro_MC | Active | | | Results Not Loaded | |
| | Two_Stage_Opa... | OpAmp | | | | | Loaded Results Out of Date | |
| | Two_Stage_Opa... | OpAmp | | | | | Results Loaded | |
| | Two_Stage_Opa... | OpAmp | maestro_adexl | Active | | | Results Not Loaded | |
| Implementation Sets | | | | | | | | |
| run_weekly | | | | | ● | | | |
| | Two_Stage_Opa... | OpAmp | maestro_MC | Active | | | | |
| run_daily | | | | | ● | | | |
| | amsPLL | PLL_VCO_320MHZ_tb | maestro | Active | | | | |

Load the simulation results

Start and stop the simulation of an individual implementation cellview.

Start the simulation of the implementation cellviews in the implementation set.

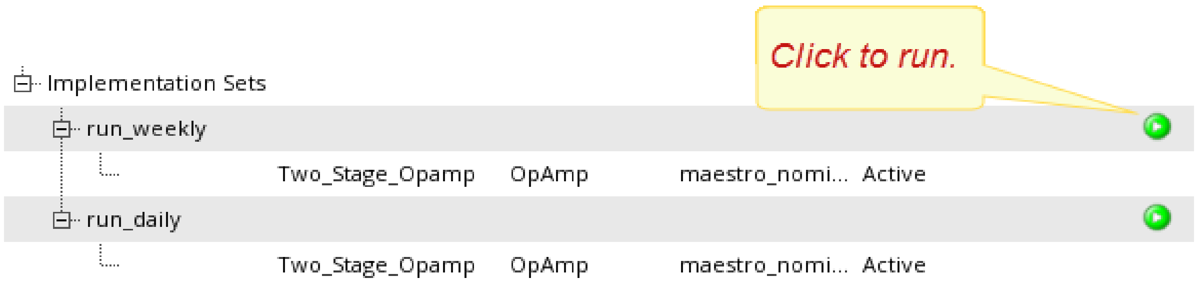
mouse L: M: R:

6(7) Open Requirement Editor

**2.** Do one of the following:

❑ Click the *Run* button corresponding to an implementation set to simulate all the cellviews organized in that set.

For this task, click *Run* corresponding to *run_weekly*.


Click to run.
Implementation Sets
run_weekly
Two_Stage_Opamp OpAmp maestro_nomi... Active
run_daily
Two_Stage_Opamp OpAmp maestro_nomi... Active

- ❑ Click the *Run* button corresponding to an individual cellview to simulate that cellview.
- ❑ Right-click an individual cellview and choose *Run Simulation*.
- ❑ Click the *Run* button on the toolbar to run all cellviews serially or in parallel. You can also set Verifier to run the cellviews incrementally in batch mode. Verifier displays text at the bottom to indicate how the cellviews will be run when you choose to use that option.
- ❑ From the *Implementations* list, select the cellviews you want to simulate and click the *Run* button on the toolbar to run the selected cellviews.

You can stop the simulations started from Verifier. For this, click the *Stop* button. You can also right-click the cellview and choose *Stop Simulation*.

*Tip* To view the requirements associated with an implementation cellview in the *Run* tab screen, select *Details*.

*Tip* You can run multiple simulations in parallel. For this, choose *Edit — Preferences*, click the *Run Options* tab in the Virtuoso ADE Verifier Preferences form, and specify the maximum number of simulations that can run in parallel in the *Maximum Implementation Jobs to Run in Parallel* field. Then click *OK*.

Consider your system capabilities to support parallel simulations when specifying this value.

*Tip* If the simulations do not run properly, check the run preferences set in Verifier, such as the default job policy.

## Task 14: Monitor Simulation Status

In Task 13, you started simulating implementation cellviews. Monitor the simulation status of the cellviews and ensure that the simulations complete successfully.

### To monitor the simulation status

➡ View the simulation run status of each cellview indicated by the corresponding status bar. The following table describes the possible status.

| Status | Description |
|---|---|
| Not Run | The cellview is not simulated, or selected for simulation. |
| Pending | The cellview has been selected for simulation and is in queue for simulation. |
| Running (50%) | The cellview is being simulated and the indicated percentage of simulation has been completed. |
| Run Finished | The cellview simulation has completed successfully. |
| Stopped | The cellview simulation was stopped manually. |
| Failed | The cellview simulation was started, but failed because of an error. When the error is encountered, a popup message displays. You can also view the details in Virtuoso CIW. |
| Run Out of Date | The cellview was simulated. However, the simulation results can be out of date because the cellview was changed since it was last simulated. The color change from green to yellow to indicate potential problems. Consider re-simulating the cellview to get the latest results. |
| Results Not Loaded | The cellview simulation results are not loaded in Verifier and Verifier cannot initiate its simulation. You must load the results from a history. |

| Status | Description |
|---|---|
| External Results Loaded | The cellview simulation results are loaded from a history and the simulation cannot be initiated from Verifier. |
| Loaded Results Out of Date | The cellview simulation results were loaded. However, these results can be out of date because the implementation cellview was changed since it was last simulated. The color change from green to yellow to indicate potential problems. Consider reloading latest results in Verifier after re-simulating the cellview externally. |

try

Attempt to resimulate an implementation cellview that has already been simulated successfully and note how Verifier treats this action.

If the cellview was not updated since the last simulation was run successfully, resimulation is not required as it would have the same results.

You can specify and review preferences in the Virtuoso ADE Verifier Preferences form.

## Task 15: View Simulation Details

Once the implementation cellviews are simulated, view their simulation results and details. Also view the log that the simulator application, such as Virtuoso Spectre Circuit Simulator, stores for each test run.

### To view simulation details, do the following as required

- Select the implementation cellview in the *Run* tab of Verifier. The right panel displays the simulation run details, such as the run mode, tests, simulation corners, run history, and other details.
- View the simulation run status and details that Virtuoso CIW and the simulation application log.
- Right-click the implementation cellview, and select *View Implementation Results*. Then, view the simulation details in the main application of the implementation cellview in read-only mode.

### To view the log that the simulator application maintains for a test run

➡ Right-click the implementation or its mapped requirement from any Verifier tab, choose *View Simulation Log of Implementation Test*, and select the test.

The simulation log of the test displays in a new window.

## Accessing Design Verification Status

This section details the following tasks.

- Task 16: Determine the Current Overall Design Verification Status
- Task 17: Review the Verification Status of Each Requirement
- Task 18: Access Detailed Verification Reports

**Note:** Verification results are stored in the `user-defined` directory by default. You can choose to store the results in the current Verifier cellview instead. For this, choose *Edit — Preferences*, click the *Run Options* tab, and set your preferred option.

### Task 16: Determine the Current Overall Design Verification Status

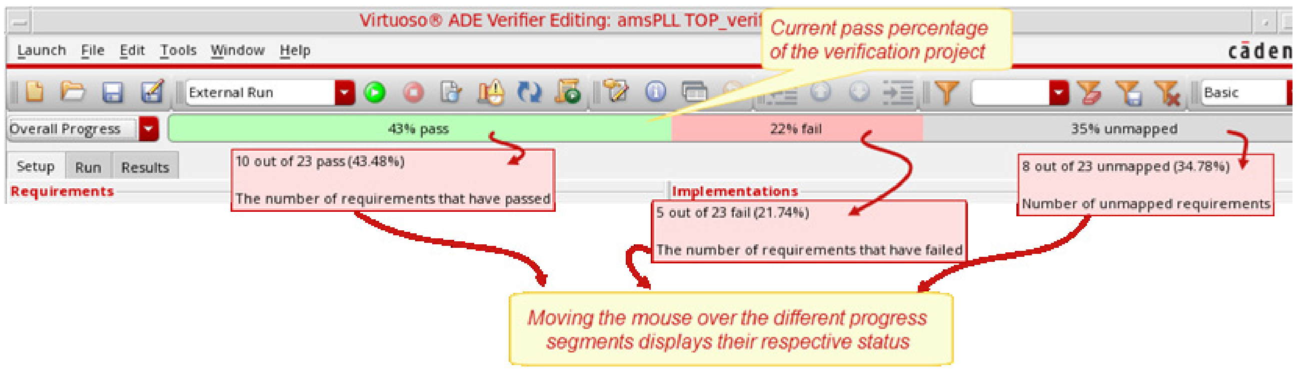
So far, you have defined the requirements and implementations for the sample design verification project and have mapped them appropriately. You have also simulated all the implementation cellviews in the project successfully. Now, determine the overall status of the requirements-driven design verification project.

### To know the overall verification status

➡ Do one of the following:

- ❑ Click the Overall Status drop-down on the toolbar.
- ❑ Choose *View—Show Status*.

The overall current verification status displays, as illustrated in the following figure. Place the cursor over *Overall Progress* to view the status snapshot.


Virtuoso® ADE Verifier Editing: amsPLL TOP_verif
Launch File Edit Tools Window Help
External Run
Basic
Overall Progress
43% pass
22% fail
35% unmapped
Setup Run Results
Requirements
Implementations
Current pass percentage of the verification project
10 out of 23 pass (43.48%)
The number of requirements that have passed
5 out of 23 fail (21.74%)
The number of requirements that have failed
8 out of 23 unmapped (34.78%)
Number of unmapped requirements
Moving the mouse over the different progress segments displays their respective status

**Note:** After the simulation of all the implementation cellviews in `run_weekly` was complete (Task 13), including `amsPLL/PLL_VCP_320MHZ_PD_td/maestro/Active`, the *Overall Progress* bar should change from 43 percent to 73 percent.

Note the following from the overall status:

- The design verification is failing currently.
- Not all requirements are mapped, and a requirement is set for manual sign off.
- All simulations were run successfully, including the simulation whose results were loaded in Verifier.
- The percentage of the verification requirements that have passed is displayed.

*Tip*

The overall verification status depends on the requirements plan and mappings, along with the results of the runs. For example, the verification plan of the sample design does not include detailed requirements for the filter and power amp blocks yet because they are planned for the future. When you add the requirements for these blocks, you will notice their contribution to the overall verification status.

## Task 17: Review the Verification Status of Each Requirement

In a requirements-driven verification flow, it is important to review the status of each requirement to identify requirements that need attention.

The failures can include requirements that are not mapped, mapped implementations that are not simulated or do not have results, failed specification checks, and where simulation results did not match the specifications.

Review the overall verification status of the requirements.

### To review the verification status of requirements

1. Click the *Results* tab.

   Verifier displays the requirements hierarchy with the verification status.

2. Review the overall verification status of each requirement.

   The following table lists the keywords and icons used to indicate the verification status of requirements.

| Icon | Keyword | Description |
|---|---|---|
| ✅ | Pass | The requirement passed verification. |
| ❌ | Fail | The requirement failed verification. |
| | Signoff Required | The requirement is of the type *Manual* and it is not signed off. |
| | Not Run | The implementation mapped to the requirement has not been run and must be run to capture simulation results. |
| ⛔ | Spec Check Fail | The specification checks failed. For details, see the corresponding requirement icon. |
| 🔗 | Not Mapped | The requirement is not mapped to any implementation. |
| ⚠ | No Results | Verifier does not have the current simulation results of the implementations associated with the requirement. |
| | Test Disabled | The test is disabled in the implementation cellview and cannot be run. |

3. Select a requirement to view its verification details in the *Result* tab or the *Information* Assistant.

*Tip*

You can filter the requirements hierarchy in the *Results* tab. For example, you can choose to view only those requirements that failed verification. For this, click the *Show* button and select the category of requirements you want to display.

Compare the specification with the corresponding simulation results to investigate any failure. You can open the results outside Verifier. For this, right-click the implementation and select *View Implementation Results*.

**Note:** Verifier is set to use the specifications set in the requirements for verification. You can check this by choosing *Edit — Preferences*. In the Virtuoso ADE Verifier Preferences form, notice that the selected option is *Verifier specifications and check to implementation*. You can then close the form.

---

try Investigate what you need to do to pass the requirement `1.4.6`, `Startup`. The verification type of this requirement is *Manual*. Therefore, you can sign off this requirement manually. See “Signing Off Failed and Manual Requirements”.

---

try Change the specification of a requirement and review its effects on the verification results.

---

try Investigate why the requirement `1.1.1.2` failed. It failed because the simulation results of the mapped implementation cellview did not meet the specifications set in the requirement.

You can take corrective actions to ensure that the requirement passes verification. For example, you right-click the implementation and choose *Edit Implementation Cellview*, edit the output details, say to `45`, and save the cellview. Then, in Verifier, right-click the implementation cellview in the *Setup* tab and choose *Overwrite Verifier Specification*.

Alternatively, if you are sure that the current results are acceptable, you can manually sign off the requirement.

---

### Task 18: Access Detailed Verification Reports

Verifier presents detailed requirements-based verification reports in text, HTML, and XML formats. These reports contain the overall verification status and verification details of each requirement, including the mapped implementations and verification results.

Access and review the verification reports.

### To view the verification report:

➡ Choose the following option from the *View* menu.

| Report Option | Description |
|---|---|
| *Publish HTML Report* | View the detailed verification status report in HTML format containing the overall status, requirement hierarchy with links, and the detailed verification status of each requirement. |

*Tip*

Verifier displays the HTML report in your default web browser.

A verification report includes the following elements:

■ Report header

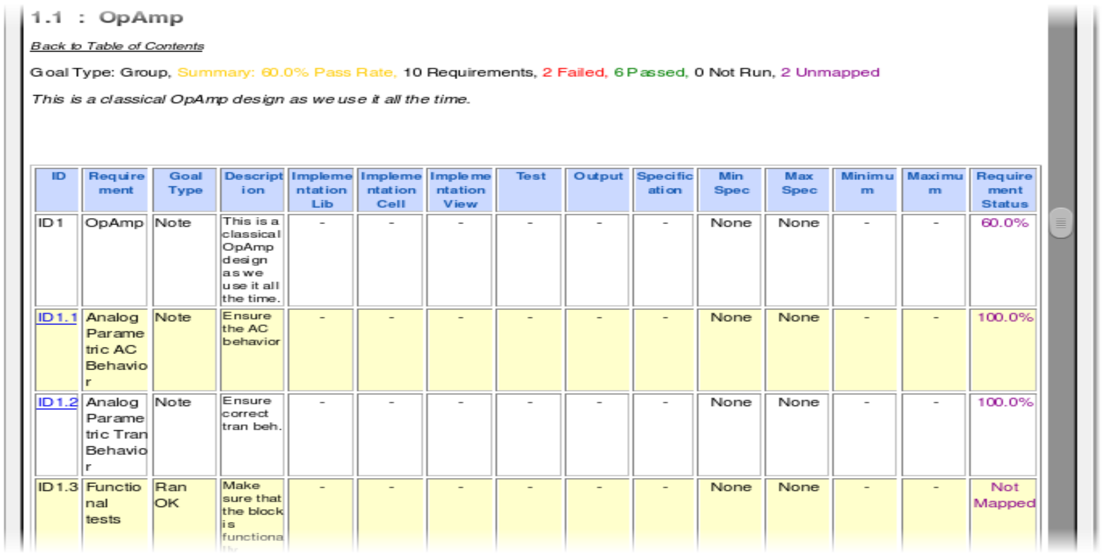
This information includes the report file name, location, creation time, and name of the creator. It also includes the overall verification status.

■ Run information

This information includes the implementation cellviews and their simulation details. The report indicates the run information of an implementation cellview.

■ Requirements-based verification details

This information includes the requirements, their verification status, and the implementation details with status. The following is an example of this information. It indicates the verification details of a parent requirement in the report in HTML format.



1.1 : OpAmp

*Back to Table of Contents*

Goal Type: Group, Summary: 60.0% Pass Rate, 10 Requirements, 2 Failed, 6 Passed, 0 Not Run, 2 Unmapped

*This is a classical OpAmp design as we use it all the time.*

| ID | Requirement | Goal Type | Description | Implementation Lib | Implementation Cell | Implementation View | Test | Output | Specification | Min Spec | Max Spec | Minimum | Maximum | Requirement Status |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ID1 | OpAmp | Note | This is a classical OpAmp design as we use it all the time. | - | - | - | - | - | - | None | None | - | - | 60.0% |
| ID1.1 | Analog Parametric AC Behavior | Note | Ensure the AC behavior | - | - | - | - | - | - | None | None | - | - | 100.0% |
| ID1.2 | Analog Parametric Tran Behavior | Note | Ensure correct tran beh. | - | - | - | - | - | - | None | None | - | - | 100.0% |
| ID1.3 | Functional tests | Ran OK | Make sure that the block is functiona | - | - | - | - | - | - | None | None | - | - | Not Mapped |

## Generating and Using Batch Scripts

This section details the following tasks.

- Task 19: Generate a Batch Script and Review the Script
- Task 20: Run the Batch Script from Verifier

Video

For a video overview of the Verifier batch scripts and how to set a cron job to run a script periodically, see *Generating and Using Batch Scripts* on Cadence Online Support.

**Task 19: Generate a Batch Script and Review the Script**

Create a batch script that you can later use to rerun the saved Verifier cellview from the command line. Review the generated script.

### To generate a batch script for the current cellview

1. Do one of the following
   - Click the *Create Batch Script* button on the toolbar.
   - Choose *Tools — Create Batch Script*.

   The Batch Script File Name form displays.
2. Specify the location and filename of the batch script.

   The default location is the current directory and the default filename is in the format `currentCell_run`. For example, `TOP_verification_run`.
3. Click *Save*.

### To review the contents of the batch script

➡ Open the generated batch script file in an editor and review the contents. The following box illustrates an example of a batch script.

```
#!/bin/csh -f
#Usage: ./<batch script file name> [<optional argument>]
#Optional argument of this batch script run:
#-run <run item> (Select <implementation set> or <lib>/<cell>/<view>/
<history> for run.)

echo "To stop this batch script run, do 'Ctrl Z', then enter 'kill -9 $$'.
This will stop Virtuoso, Verifier and the simulations also."
echo "Using 'Ctrl C' will stop the batch script but Virtuoso, Verifier and
the simulations will continue to run."

startBatchVerifier $* -scriptid $$ -lib amsPLL -cell TOP_verification -view
verification -readPermission  -rep ./TOP_verification.html -log
TOP_verification_run.log
```

### Task 20: Run the Batch Script from Verifier

Run the batch script you created in Task 19 from the command line. Also monitor the progress from the graphical user interface of Verifier.

try Before you run the batch script, set your preferences in Verifier. For example, you can choose to hide or display runtime status information.

### To run the batch script

➡ Run the script from the command line.

For example:

```
$ ./TOP_verification_run
```

Verifier runs the batch script. You can monitor the status of the batch run in Verifier.

**Note:** When the batch script run is completed, it returns `0` for pass status and `1` for fail status. For example, the following return value indicates that the run was successfully completed:

```
$ echo ?
$ 0
```

*Important*

To stop Verifier and the simulations with the correct return status, press `Ctrl+Z` and then kill the batch script process using the `kill -9` *`PID`* command. Do not attempt to stop the batch script by pressing `Ctrl+C` because this action will leave Verifier and the simulations running.

3

# Using Advanced Verifier Features

This chapter demonstrates the advanced features of Virtuoso ADE Verifier (Verifier) that are not described in Chapter 2, “Performing Basic Verification Flow Tasks.”

This chapter includes the following topics:

## Exporting and Including Requirements

This section details the following tasks.

- Task 1: Export Requirements
- Task 2: Include Requirements by Reference or Copy

### Task 1: Export Requirements

You can export the requirements hierarchy set in a Verifier session, also referred to as a verification plan, to a comma-separated values, `.csv`, file. You can store the exported verification plan for future reference and reuse it in other verification projects.

Open the cellview `amsPLL/TOP_verification/verification` and export the verification plan stored in this cellview to a new file called `myPlan.csv`.

**To export the verification plan:**

1. Choose *File — Export — CSV*.

   The *Export Requirements to CSV File* form displays.

2. Specify the name and location of the `.csv` file.
3. Specify the details to export.
4. Click *Save*.

Verifier exports the requirements hierarchy to the specified file.

try

Open `myPlan.csv` in an editor and review how Verifier saved the verification plan. Note the order of the fields. The following snippet from `myPlan.csv` indicates the fields and their order, along with an example requirement.

```
Parent,ID,Title,Type,MinSpec,MaxSpec,Unit,Owner,Description,Overall
Status,Mapping,History,Result Data Age,Min Value,Typical Value,Max Value,
Passed,Failed,No Results,Unmapped,Verification Space,Coverage,
AnVerProjTop,"Analog Verification Project",note,,,,Tom,
```

The hierarchical numbers are not included in the `.csv` file. These numbers are auto-generated by Verifier.

## Task 2: Include Requirements by Reference or Copy

**Note:** Before you perform this task, save and close the cellview `amsPLL/TOP_verification/verification`.

You can include requirements in Verifier from the following sources:

- Spreadsheet (`.xlsx`, `.XLSX`)
- Comma-separated values stored in a file (`.csv`, `.CSV`)
- Cellview of the type `verifier`

**Note:** The `.xls` or `.XLS` Excel file format is not supported.

Including requirements is typically the first step in a top-down verification flow.

For information on adding a requirement manually, see Task 4: Add and Move a Requirement in Chapter 2, "Performing Basic Verification Flow Tasks." For information on creating a requirement from an implementation, see Task 6 in this chapter.

This task includes the following parts:

*Part 1*: Start a new Verifier session and include the requirements stored in `myPlan.csv` created in Task 1 by reference.

*Part 2*: Save the cellview as `amsPLL/TOP_verification/myVerifierCellview`.

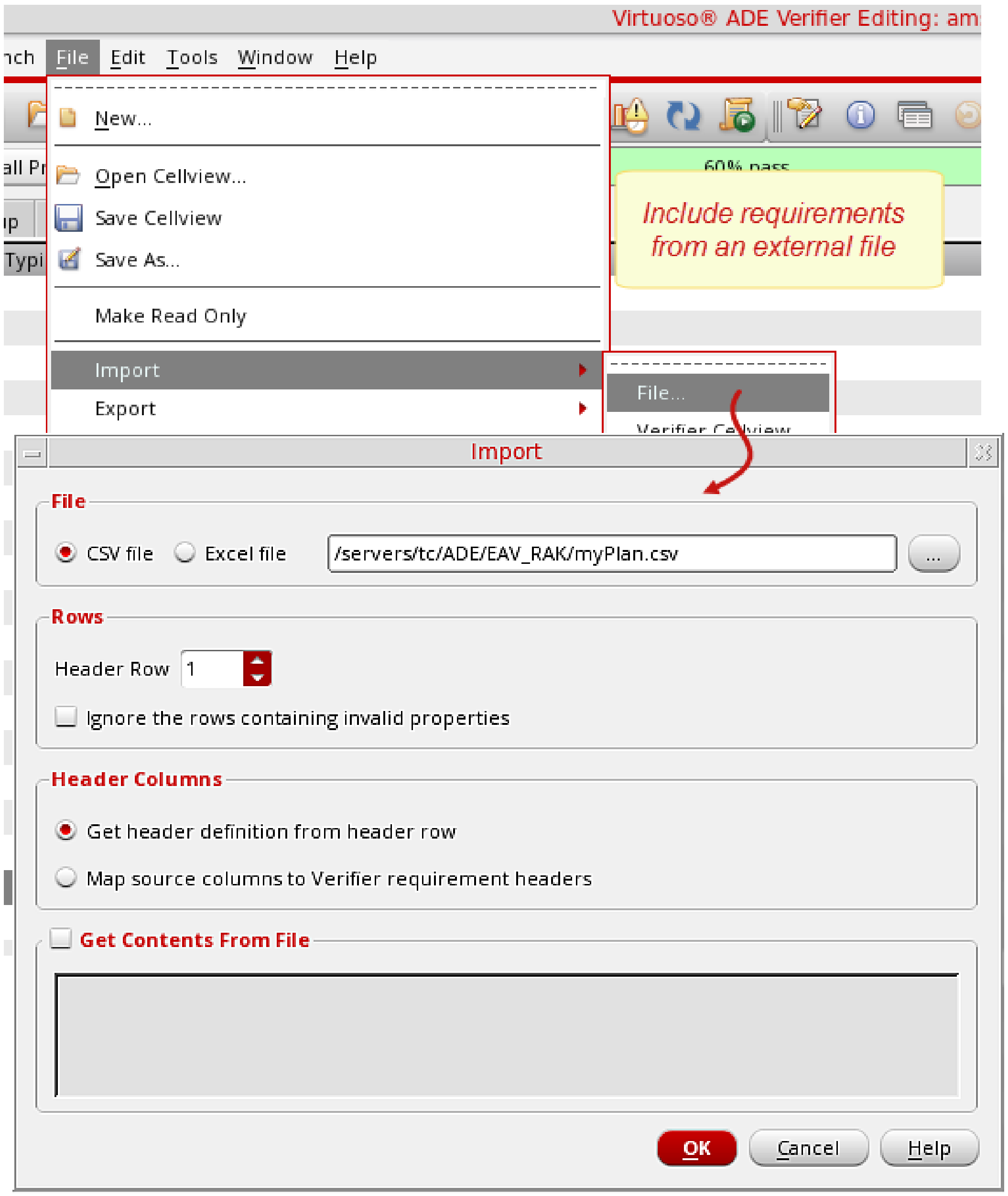
*Part 3*: Remove the referenced requirements from the cellview.

*Part 4*: Include the same requirements in `myPlan.csv` by copying them in the cellview.

*Part 5*: Analyze the difference between the referenced and copied requirements.

### *Part 1:* To include requirements from by copying them in a Verifier session:

1. Choose *Tools — Import* and select *CSV* or *Excel*. The Include form displays.
2. Specify the required information as described in the procedure to include requirements from an external source. Ensure that you choose *Copy* in the *Operation* group box.
3. Click *OK*.



**File**
(● CSV file) ○ Excel file — /servers/tc/ADE/EAV_RAK/myPlan.csv

**Rows**
Header Row: 1
☐ Ignore the rows containing invalid properties

**Header Columns**
(● Get header definition from header row)
○ Map source columns to Verifier requirement headers

☐ **Get Contents From File**

**4.** Select the source type.

For this task, select *CSV File*.

**5.** Specify the source.

For this task, specify the `myPlan.csv` and its location in the *File Name* field.

**Note:** The sample design database includes some `.xlsx` and `.csv` files in the `Verifier_Example_Files` directory. You can use these files as the sources of the requirement plan.

**6.** If the source is a CSV file or a Microsoft Excel file, do the following:

**a.** If your source is a Microsoft Excel file with multiple sheets, specify the sheet that contains the requirements you want to import. For this, click *Get Sheets* and select the sheet name or *All*, as required.

Try working with `amsPLL_TOP_verification2.xlsx` in the `Verifier_Example_Files` directory.

**b.** In the *Number of Rows to Be Ignored* group box, specify the top and bottom rows that must be ignored. For example, if the source file includes column headers in the first row that Verifier must ignore when including requirements, specify `1` in the *Top rows* field.

**c.** Set the *Header Columns* group box as described below.

To use the requirement column definitions from the source file, click *Get header definition from row* and specify the header row number. Ensure that the headers in the source use the following text. The mandatory header is `Title`

```
Parent,ID,Title,Type,MinSpec,MaxSpec,Unit,Owner,Description
```

For this task, click *Get header definition from row*.

To map the source columns to the Verifier requirement columns, do the following.

**1.** Click *Map source columns to Verifier requirement headers*.

**2.** In the *Number of Columns to be Loaded* group box, specify the number of columns you want to import. The column mapping table in the *Header Columns* group box updates accordingly.

**3.** Map the columns in the source file to the Verifier requirement columns. You can change the sequence of the columns based on the sequence in which they exist in the source file. For example, if the third column in the source file is the description instead of the title, select *Description* under *3*.

You can merge multiple cells by assigning the same target to multiple columns. For example, if the columns 10 and 11 of the file contain information that you want to add as the description of the included requirements, assign `Description` to both these columns.

**Note:** The hierarchy number (*Hier*) is auto-assigned by Verifier and is not part of the import process.

**Note:** It is possible that the source file does not have separate parent requirement and ID columns. Instead, the file can use a path-type ID, such as `AVP.OPAMP` to indicate that `AVP` is the parent of `OPAMP`. In this case, ignore the `Parent` and `ID` mapping and set `idHierDelimiter` to specify the delimiter character to determine the hierarchy. Verifier uses this delimiter to determine the hierarchy level and the ID.

**d.** Select *Get Column From File* to view some rows in the source file to ensure that you have specified the correct file and mapping details. The number of rows that will be shown in this field is *Top rows* + `1`.

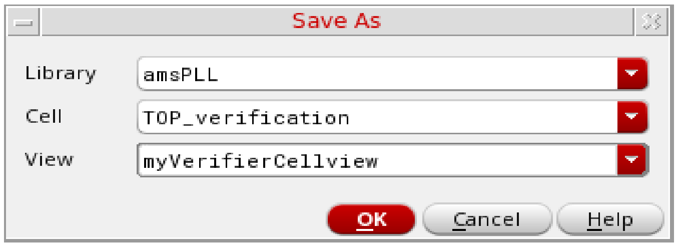
**7.** Click *OK*.

The specified data is included in the Verifier session. The first node in the requirements hierarchy represents the source file.

For this task, you must include the requirements stored in `myPlan.csv` by reference. Now analyze how you can work with the requirements included by reference. Note down your observations.

### *Part 2:* Save the cellview

1. Click *File — Save As* or click the *Save As* icon on the toolbar.
2. Specify the library, cell, and view. For this task, specify `amsPLL/TOP_verification/myVerifierCellview`.
3. Click *OK*.



| | |
|---|---|
| Library | amsPLL |
| Cell | TOP_verification |
| View | myVerifierCellview |

### *Part 3:* Remove the requirements you included by references

1. Right-click the referenced requirement and click *Delete Requirements*.
2. The referenced requirements are removed from the cellview.

### *Part 4:* Include the requirements by referencing them in the cellview

1. Choose *Tools — ADE Verifier* from Virtuoso CIW window.

   The form *Choose Verifier Cellview* for the Verifier Setup displays.
2. Specify the required details.
3. The cellview opens.
4. Choose *File — Import — Verifier Cellview*.

   The *Import Verifier Cellview* form displays.
5. Specify the import operation.

   ❑ Click *Copy* if you want to include the requirements by copying them in the Verifier setup. The requirements will be copied in edit mode, without any links to the source.

   ❑ Click *Reference* if you want to include the requirements by reference. The requirements will be added as read-only entries. You must maintain the source to ensure that the reference links do not break.

   For this task, click *Reference*.
6. Click *OK*.

   The external cellview is added to the current cellview as a reference with the type *ExtRef.*

### *Part 5:* Analyze the difference between the requirements included by reference and copy

The following table describes some observations.

| Type | Referenced Requirements | Copied Requirements |
|---|---|---|
| Edit | Cannot edit in Verifier.<br><br>Edit the requirements in the source file and update data available to Verifier. | Can edit directly in the requirements hierarchy and using requirement editor.<br><br>Because the copied requirements have no link with the source, changes in the source are not reflected in Verifier. |
| Delete | Cannot delete requirements in Verifier.<br><br>Delete the requirement in the source file and update data available to Verifier. | Can delete from the requirements hierarchy. |
| Mapping | Cannot map with implementations.<br><br>Can highlight requirement–implementation mappings, if the mapped requirement and implementation are visible.<br><br>Edit or delete the mappings in the source file and update data available to Verifier.<br><br>**Note:** If the referenced requirements are mapped elsewhere, as done in designer cellviews, then their mapping cannot be changed locally. For details, see “Setting Requirements for External References”. | Can map with implementations that are not referenced.<br><br>Can remove the mappings if implementations are not referenced.<br><br>Can highlight requirement–implementation mappings. |
| Add More | Cannot add additional requirements manually or from a CSV or Excel file. You cannot add requirements from a cellview.<br><br>**Note:** You cannot add new requirements within a referenced requirement hierarchy. You can add requirements above or below the referenced hierarchy. | Can add additional requirements manually or from a CSV or Excel sources.<br><br>You cannot add referenced requirements from a cellview. |

try

Currently, the new cellview `amsPLL/TOP_verification/myVerifierCellview` has the requirements copied from `myPlan.csv`. Save this cellview. Now attempt to import the same requirements in `myPlan.csv` again. What do you observe?

Verifier does not let you add duplicate requirements. The requirement IDs must be unique.

## Setting Custom Fields

This section details the following tasks:

- Task 3: Prepare the Verifier Environment to Use Custom Fields
- Task 4: Use the Custom Fields

### Task 3: Prepare the Verifier Environment to Use Custom Fields

Verifier lets you include additional information for your design verification project, and for each requirement in the project. For example, you can include the name and e-mail address of the verification project manager. For each requirement in the project, you can include the name and e-mail address of the engineer.

To use custom fields, you must define them in a `.csv` file and set the Verifier environment to use those fields. In this task, prepare the Verifier environment to use custom fields.

*Tip*

Before you set Verifier to start using your custom fields, determine if you want different custom fields for the project and its requirements. This task considers that you want different custom fields. Therefore, the task uses two `.csv` files, one for the custom project fields and one for the custom requirement fields. If your custom fields are common for the project and its requirements, you can use a single `.csv` file.

**To prepare the Verifier environment for incorporating your custom fields**

1. Specify the custom field tags, names, and tool tips in the `.csv` files for the project and requirements. Use the following format:

```
Field tag name,Field tag description,Field tool tip
```

See the following examples:

The file `Verifier_Example_Files/infoProject.csv` in the sample design database contains the following custom field details for the verification project:

```
VerificationManager,Verification Manager,The name of the verification project
manager.
Email,E-mail Address,The e-mail address of the project manager.
```

The file `Verifier_Example_Files/infoReq.csv` in the sample design database contains the following custom field details for the requirements:

```
Typical,Typical Value, The typcial value for the parameter.
Engineer,Engineer Name,The name of the lead engineer.
Email,E-mail Address,The e-mail address of the engineer.
```

2. Set the Verifier environment variable `customFieldConfig` and `customReqFieldConfig` to indicate the `.csv` files of the custom project fields and the custom requirement fields, respectively.

   You can load `setCustomField.il` in the `Verifier_Example_Files` directory to set the variables. For this, run the following command in Virtuoso CIW.

   ```
   load("./Verifier_Example_Files/setCustomField.il")
   ```

   Alternatively, you can set the variables in the `.cdsenv` or `.cdsinit` file, or from Virtuoso CIW. The following snippet illustrates how you set these variables in the `.cdsinit` file stored in the root directory of your project.

   The filename and location entry of the `.csv` files can contain the character ~ for user home directory, and *`$UNIX_environment_variable`* for the value of an environment variable.

   ```
   envSetVal("verifier.customFieldConfig" "csvfile" 'string "~/infoProject.csv")
   envSetVal("verifier.customReqFieldConfig" "csvfile" 'string "~/infoReq.csv")
   ```

## Notes:

■ After you have prepared the Verifier environment, the custom fields become available to the Verifier sessions. The custom project fields are available in the Virtuoso ADE Verifier Preferences form. The custom requirement fields are available in the requirement editor.

- You set the value of the custom fields in a Verifier session. These values are saved in the Verifier cellviews.

### Task 4: Use the Custom Fields

Reopen the cellview `amsPLL/TOP_verification/myVerificationCellview` and set the values of the custom project fields and the custom requirement fields.

### To set the values of the custom project fields

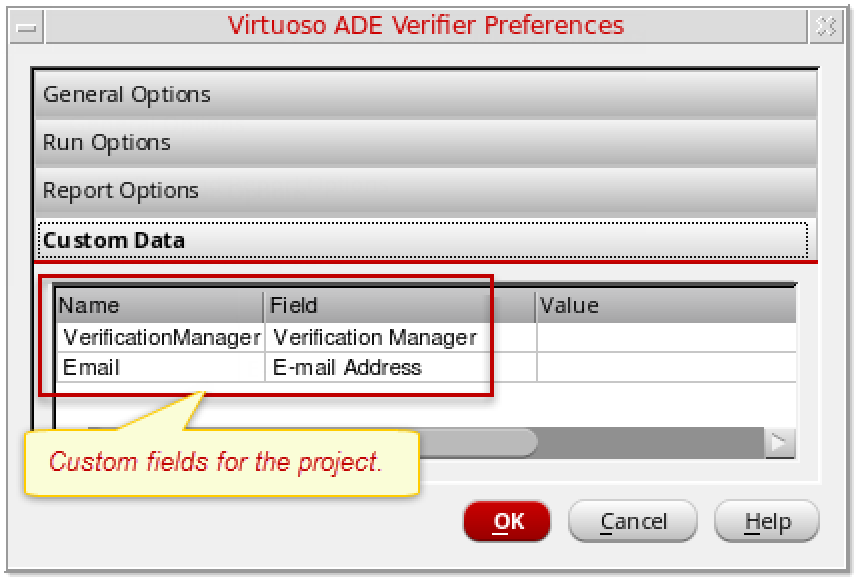
1. Choose *Edit — Preferences*.

   The Virtuoso ADE Verifier Preferences form displays.

2. Click the *Custom Data* tab.

   The custom fields display in a table.

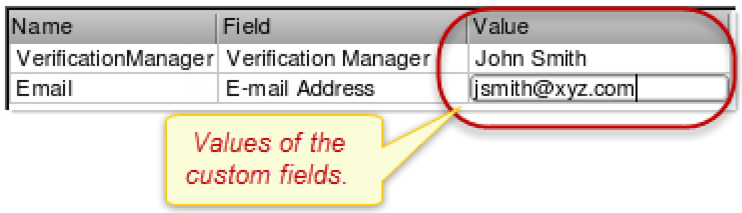
   **Note:** The *Custom Data* tab becomes available when you set the Verifier environment variable `customFieldConfig`.


Virtuoso ADE Verifier Preferences
General Options
Run Options
Report Options
Custom Data
Custom fields for the project.
OK
Cancel
Help

| Name | Field | Value |
|---|---|---|
| VerificationManager | Verification Manager | |
| Email | E-mail Address | |

3. Double-click the *Value* cell of a custom field and type the value.

Repeat this step to specify the value of the remaining custom fields.



| Name | Field | Value |
|---|---|---|
| VerificationManager | Verification Manager | John Smith |
| Email | E-mail Address | jsmith@xyz.com |

4. Click *OK*.

You can save your changes in the cellview.

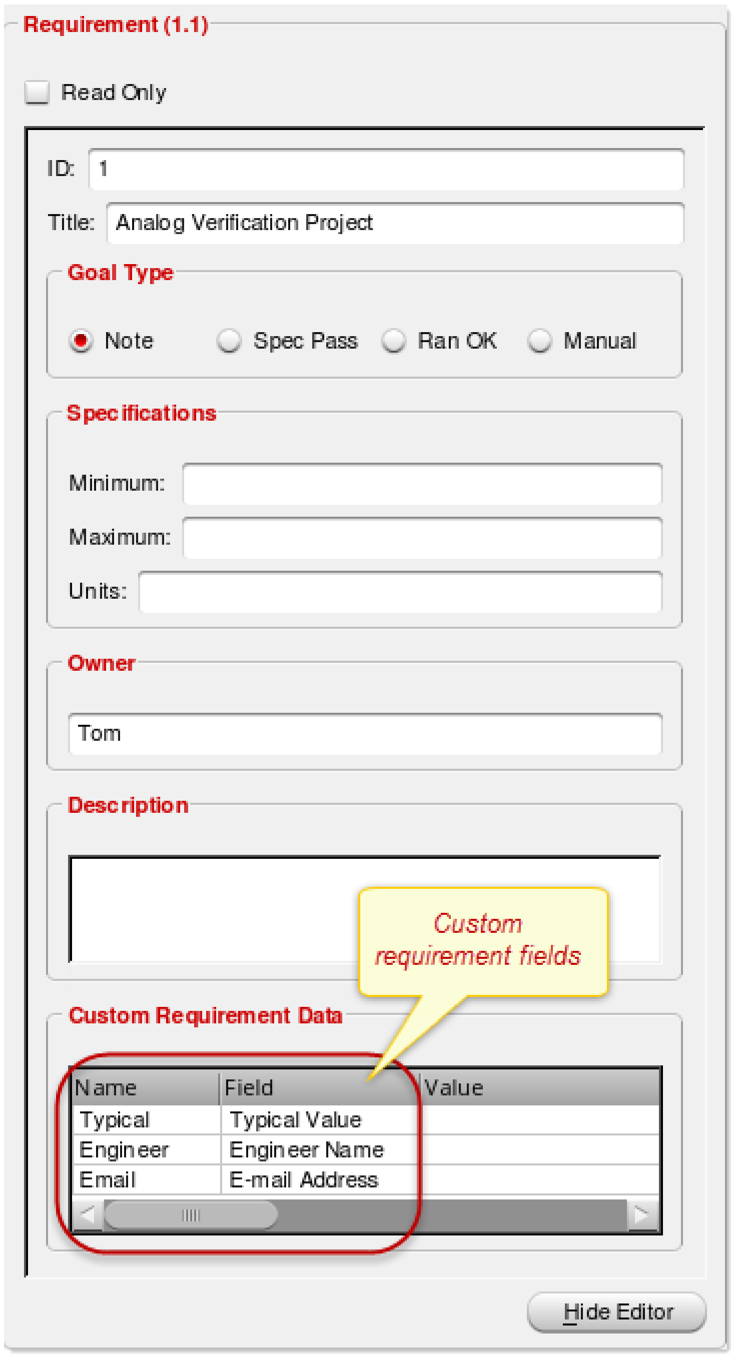
**To set the values of the custom fields of a requirement**

1. Click the *Setup* tab, if it is not the current tab.
2. Select the requirement.
3. Click the requirement editor button on the toolbar. Alternatively, choose *Edit — Open Requirements Editor*.

   The requirement editor displays the custom fields in the *Custom Requirement Data* area, as illustrated in the following figure.

   **Note:** The *Custom Requirement Data* area becomes available when you set the

Verifier environment variable `customReqFieldConfig`.


Requirement (1.1)

Read Only

ID: 1

Title: Analog Verification Project

Goal Type

[✓ Note] Spec Pass Ran OK Manual

Specifications

Minimum:

Maximum:

Units:

Owner

Tom

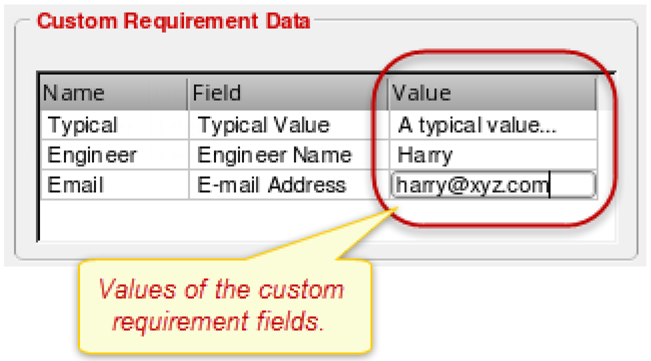
Description

Custom Requirement Data

| Name | Field | Value |
|---|---|---|
| Typical | Typical Value | |
| Engineer | Engineer Name | |
| Email | E-mail Address | |

**4.** Double-click the *Value* cell of a custom field and type the value.

Repeat this step to specify the value of the remaining custom fields.



| Name | Field | Value |
|---|---|---|
| Typical | Typical Value | A typical value... |
| Engineer | Engineer Name | Harry |
| Email | E-mail Address | harry@xyz.com |

You can save your changes in the cellview.

## Including Implementations and Adding their Requirements

This section details the following tasks:

- Task 5: Include Implementations from Other Cellviews
- Task 6: Create Requirements from Implementations

### Task 5: Include Implementations from Other Cellviews

You can add implementations using the following methods:

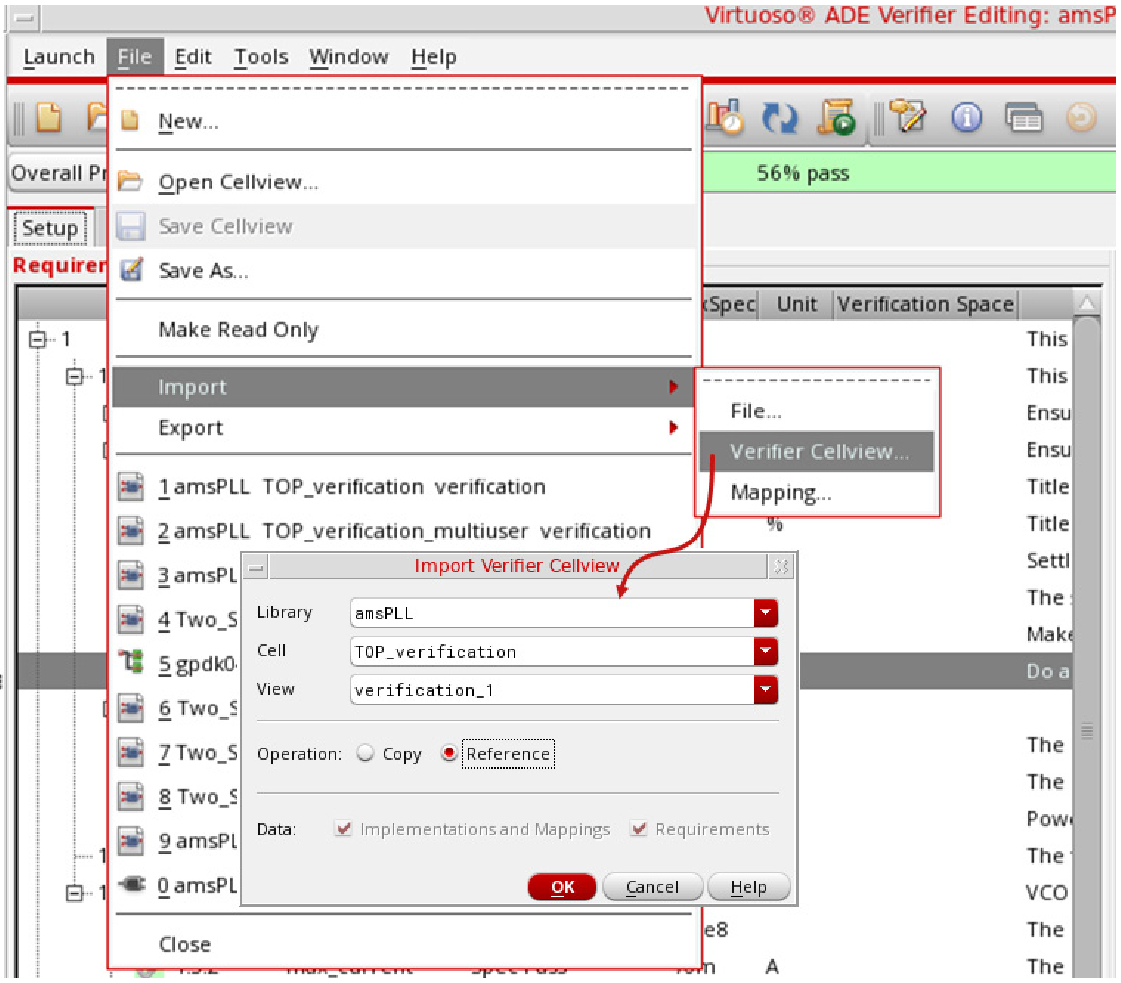
- Add implementations manually.
- Copy or reference implementations and their mappings defined in another cellview of the type `verifier`.
- Add multiple implementation cellviews using a Verifier utility.

In the cellview `amsPLL/TOP_verification/myVerificationCellview`, copy the implementations present in the Verifier cellview `amsPLL/TOP_verification/Verification`.

Also practice including multiple implementations. You use this method when entering a bottom-up verification flow.

## To include the implementations of another cellview in the current setup

1. Choose *File — Import — Verifier Cellview*. The Import Verifier Cellview form displays.



Library: amsPLL
Cell: TOP_verification
View: verification_1
Operation: Copy [✓ Reference]
Data: [✓ Implementations and Mappings] [✓ Requirements]

2. Select the source cellview in the *Cellview* area.
3. Click *Copy* to copy the data from the source cellview.

   Click *Reference* to include the implementations by reference.
4. Select *Implementations and Mappings*.

   If you want to include the requirement from the source cellview, select *Requirements*.

   **Note:** Verifier also includes the requirement–implementation mappings defined in the source cellview if the corresponding requirement IDs exist in the current cellview.
5. Click *OK*.

The implementations are included in the current Verifier session. The mapping information is passed on with the included implementations in the destination cellview, provided the corresponding requirements exists. You can save the cellview.

### To include multiple implementations in a Verifier setup

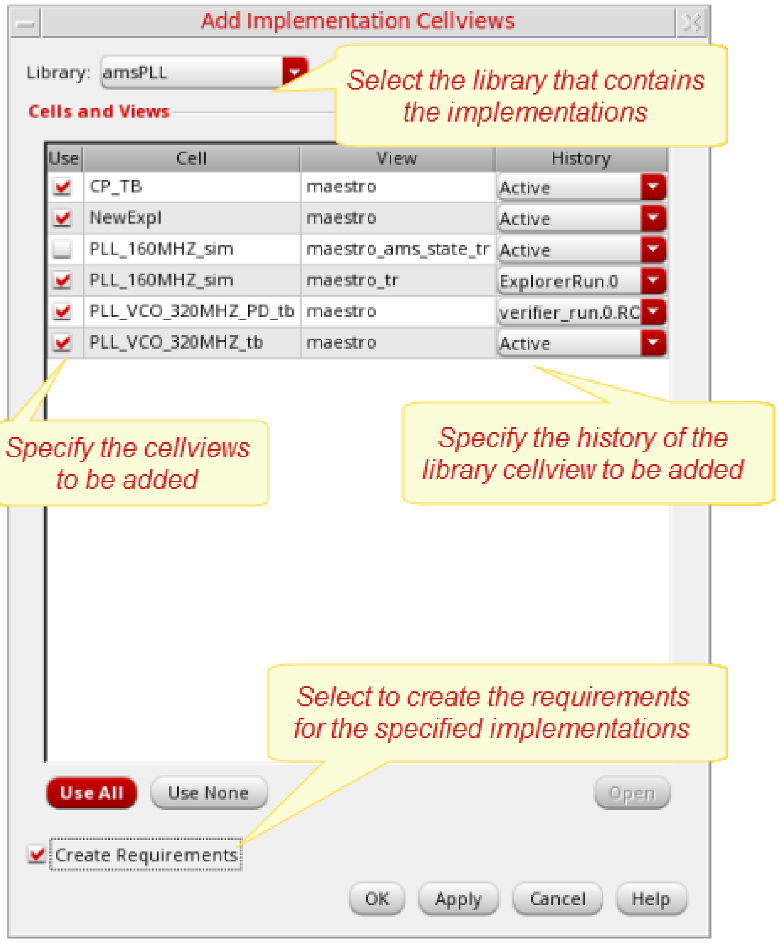
1. Choose *Tools — ADE Verifier*.

   The form Load Initial Data for the New Verifier Setup displays.

2. Choose *Tools — Add Implementation Cellviews*.

   The Add Implementation Cellviews form displays.

3. Specify the necessary information to add multiple implementations:


Add Implementation Cellviews
Select the library that contains the implementations
Cells and Views
Specify the cellviews to be added
Specify the history of the library cellview to be added
Select to create the requirements for the specified implementations
Use All
Use None
Open
OK
Apply
Cancel
Help

Library: amsPLL

| Use | Cell | View | History |
|---|---|---|---|
| ✓ | CP_TB | maestro | Active |
| ✓ | NewExpl | maestro | Active |
|  | PLL_160MHZ_sim | maestro_ams_state_tr | Active |
| ✓ | PLL_160MHZ_sim | maestro_tr | ExplorerRun.0 |
| ✓ | PLL_VCO_320MHZ_PD_tb | maestro | verifier_run.0.RC |
| ✓ | PLL_VCO_320MHZ_tb | maestro | Active |

✓ Create Requirements

   a. Select the library that contains the implementation cellviews.

   b. Select the cellviews you want to add as implementation cellviews.

   c. Select *Create Requirements* to create the requirements corresponding to the implementations. The implementations are also mapped with the corresponding requirements. You can specify the tests and outputs for which requirements must be created.

4. Click *OK*.

Based on the specified information, Verifier adds the implementations, and optionally mapped auto-created requirements.

### Task 6: Create Requirements from Implementations

In the previous task, you included multiple implementations and created their corresponding requirements. In this task, delete the requirement mapped with a specific implementation. Then, create a requirement for that implementation.

try Instead of deleting the requirement, try removing the mapping. For this, right-click the requirement or implementation and choose *Delete Mapping*.

## To create a requirement from an implementation

1. Right-click the implementation that is not mapped.
2. Choose *Create Requirements*.

Verifier created a requirement mapped with the implementation at the end of the requirements hierarchy. You can change the hierarchical position of the new requirement as needed.

try You can create requirements from multiple implementations. For this, select multiple implementations, right-click, and choose *Create Requirements*.

After investigating the feature to add multiple implementations and creating requirements from implementations, you can close the Verifier setup without saving it.

## Overriding the Job Policy

**Task 7: Overriding the Job Policy**

Verifier uses the default job policy of the implementation cellview application for running the implementation cellview simulations.

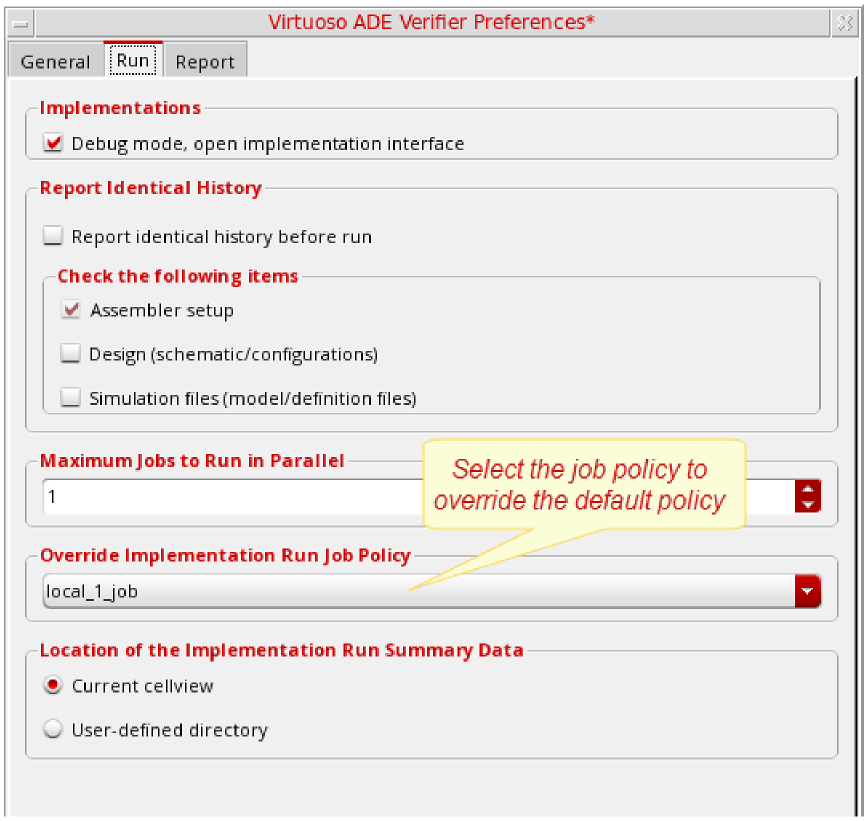
Override the job policy in Verifier globally.

### To override the default job policy

1. Choose *Edit — Preferences*.

   The Virtuoso ADE Verifier Preferences form displays.

2. Click the *Run Options* tab.


Virtuoso ADE Verifier Preferences*
General
Run
Report
Select the job policy to override the default policy

Implementations
- [✓] Debug mode, open implementation interface

Report Identical History
- [ ] Report identical history before run

Check the following items
- [✓] Assembler setup
- [ ] Design (schematic/configurations)
- [ ] Simulation files (model/definition files)

Maximum Jobs to Run in Parallel: 1

Override Implementation Run Job Policy: local_1_job

Location of the Implementation Run Summary Data
- ● Current cellview
- ○ User-defined directory

3. Select the job policy you want to use from the drop-down list in the *Override Implementation Run Job Policy* area.

4. Click *OK*.

*Tip*

You can define the job policy in ADE Assembler and ADE Explorer and select it in the Virtuoso ADE Verifier Preferences form.

try Open the Virtuoso ADE Verifier Preferences form and explore other options that you can set globally.

## Using Monte Carlo Sampling

**Task 8: How Verifier Manages Monte Carlo Sampling Runs**

Verifier supports the *Monte Carlo Sampling* and *Single Runs, Sweeps and Corners* run modes.

In `amsPLL/TOP_verification/verification`, requirement `1.1.5.2` is mapped to the following implementation whose run mode is *Monte Carlo Sampling*:

```
Two_Stage_Opamp/OpAmp/maestro_MC/Active.AC::Overall_Yield
```

Observe the outputs of this implementation cellview in Verifier. Then, change the run mode of the implementation cellview and review the changes in the outputs. Reset the run mode to Monte Carlo sampling.

### Observe the outputs of an implementation cellview whose run mode is Monte Carlo Sampling

➡ When an implementation cellview is set to use the *Monte Carlo Sampling* run mode, the following sections display in Verifier:

- ❑ *Output Values*: This section displays the outputs, with values, of the tests in the implementation cellview.
- ❑ *Statistical Values*: This section displays the yield, mean, standard deviation, and CPK of each output measurement of the tests in the implementation cellview. It also shows the overall yield of each test.

You can map Monte Carlo outputs and statistical values with requirements. Ensure that the requirements that are mapped to Monte Carlo statistical values have specifications. Otherwise, the specification checks fail.

*Important*

The specification values used for Monte Carlo statistical parameters are different from the output specification values in ADE Assembler or ADE Explorer. For example, consider that you have a specification value of `DCGain` as `>10` in ADE Assembler. Your Monte Carlo simulation can provide you a yield measurement of `98%` for `DCGain`. The specification value specified in Verifier for such statistical parameters reference the statistical values. This means that a specification value of minimum `0.97` for the requirement mapped to `DCGain::Yield` will ensure that the requirement passes.

### To change the run mode of an implementation cellview and observe the changes in outputs

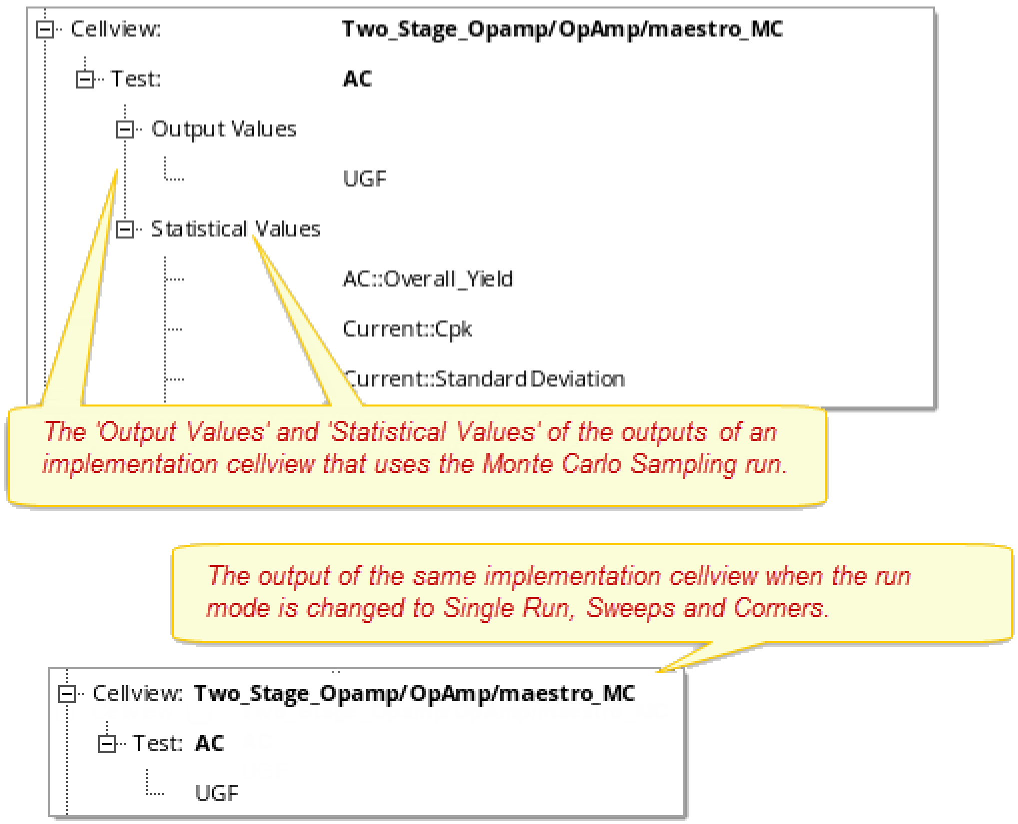
1. Right-click the implementation cellview and choose *Edit Implementation Cellview*.

   The cellview opens in ADE Assembler.

2. Change the run mode to *Single Run, Sweeps and Corners*, save and close the cellview in ADE Assembler.

3. Observe the changes in the outputs in Verifier.

Verifier now shows the outputs relevant to the current run mode. Also, the mappings are removed.


Cellview: Two_Stage_Opamp/OpAmp/maestro_MC
Test: AC
Output Values
UGF
Statistical Values
AC::Overall_Yield
Current::Cpk
Current::Standard Deviation
The 'Output Values' and 'Statistical Values' of the outputs of an implementation cellview that uses the Monte Carlo Sampling run.
The output of the same implementation cellview when the run mode is changed to Single Run, Sweeps and Corners.
Cellview: Two_Stage_Opamp/OpAmp/maestro_MC
Test: AC
UGF

Reset the run mode to *Monte Carlo Sampling* and restore the mappings. You can manually map the requirements with the statistical values and outputs.

**Note:** You can start the *Single Run, Sweeps and Corners* or the *Monte Carlo Sampling* run of an implementation cellview from Verifier. It is also possible to run the implementation cellview outside Verifier and load the results in Verifier. For Monte Carlo run mode, Verifier can retrieve the statistical values from the relevant history of the cellview.

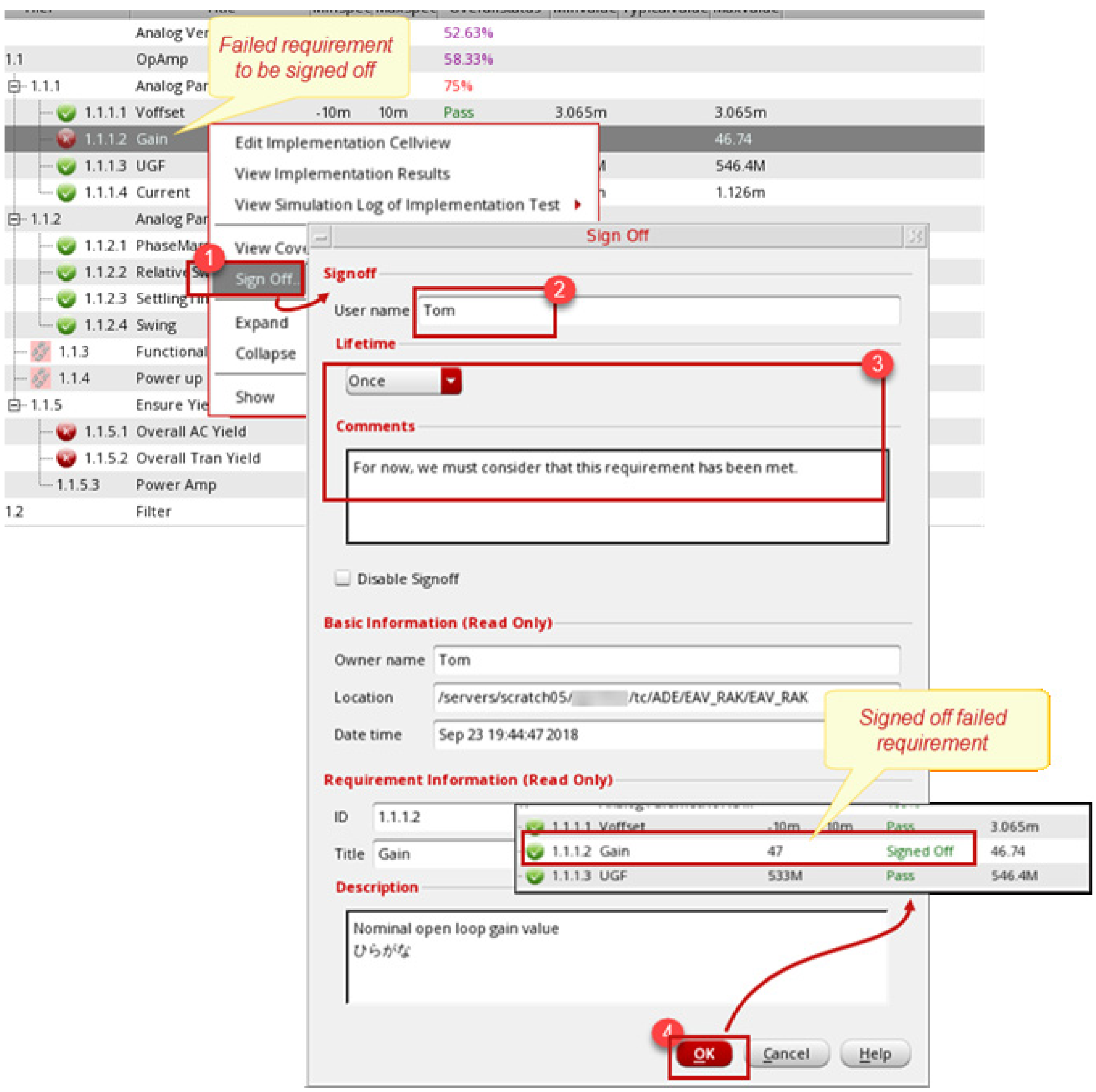
## Signing Off Failed and Manual Requirements

This section details the following tasks.

- Task 9: Sign Off a Failed Requirement
- Task 10: Sign Off a Requirement of the Type Manual

## Task 9: Sign Off a Failed Requirement

The verification type of `1.1.1.2` in `amsPLL/TOP_verification/verification` failed verification. Consider that this failure is acceptable. To highlight this in your verification plan, perform a manual signoff.


Failed requirement to be signed off
Edit Implementation Cellview
View Implementation Results
View Simulation Log of Implementation Test
View Cove
Sign Off...
Expand
Collapse
Show
1
2
3
4
Signed off failed requirement
OK
Cancel
Help

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | Analog Ver | | | 52.63% | | | |
| 1.1 | OpAmp | | | 58.33% | | | |
| 1.1.1 | Analog Par | | | 75% | | | |
| 1.1.1.1 | Voffset | -10m | 10m | Pass | 3.065m | | 3.065m |
| 1.1.1.2 | Gain | | | | | | 46.74 |
| 1.1.1.3 | UGF | | | | | | 546.4M |
| 1.1.1.4 | Current | | | | | | 1.126m |
| 1.1.2 | Analog Par | | | | | | |
| 1.1.2.1 | PhaseMar | | | | | | |
| 1.1.2.2 | Relative | | | | | | |
| 1.1.2.3 | Settling | | | | | | |
| 1.1.2.4 | Swing | | | | | | |
| 1.1.3 | Functional | | | | | | |
| 1.1.4 | Power up | | | | | | |
| 1.1.5 | Ensure Yie | | | | | | |
| 1.1.5.1 | Overall AC Yield | | | | | | |
| 1.1.5.2 | Overall Tran Yield | | | | | | |
| 1.1.5.3 | Power Amp | | | | | | |
| 1.2 | Filter | | | | | | |

**Sign Off**

**Signoff**

User name: Tom

**Lifetime**

Once

**Comments**

For now, we must consider that this requirement has been met.

☐ Disable Signoff

**Basic Information (Read Only)**

Owner name: Tom
Location: /servers/scratch05/[redacted]/tc/ADE/EAV_RAK/EAV_RAK
Date time: Sep 23 19:44:47 2018

**Requirement Information (Read Only)**

ID: 1.1.1.2
Title: Gain

**Description**

Nominal open loop gain value
ひらがな

| | | | | | | |
|---|---|---|---|---|---|---|
| 1.1.1.1 | Voffset | -10m | 10m | Pass | 3.065m | |
| 1.1.1.2 | Gain | 47 | | Signed Off | 46.74 | |
| 1.1.1.3 | UGF | 533M | | Pass | 546.4M | |

### To sign off a failed requirement

1. In the *Results* tab, right-click the failed requirement and select *Sign off*.

   The Sign Off form displays.

2. Type a name in the *User Name* field.

3. Type details of the signoff, such as the reason, in the *Comments* field.

   You can specify the valid period of the signoff from the *Lifetime of Signoff* drop-down list. The available options are:

   ❑ *Once*: The signoff is valid only in the current Verifier session.

   ❑ *Indefinitely*: The signoff is valid for an infinite time.

   ❑ *Until Date*: The signoff is valid up to the date specified in the corresponding field.

4. Click *OK*.

   The status of the requirement changes to *Signed Off*.

try After signing off a requirement manually, select it and review its details in the *Information* area of the *Results* tab screen. Also, note the increment in the overall progress percentage.

**Task 10: Sign Off a Requirement of the Type *Manual***

The verification type of `1.4.6` in `amsPLL/TOP_verification/verification` is set as *Manual*. Mark this requirement as verified and passed.

**Note:** The procedure to sign off a requirement of the goal type *Manual* is similar to signing off a failed requirement.

### To sign off a requirement whose goal type is Manual

1. In the *Results* tab, right-click the requirement and select *Manual Signoff*.

   The Verifier Signoff form displays.

2. Type a descriptive signoff title in the *Name* field.

   This name is populated in the *Owner* field.

3. Specify the lifetime of the signoff.

4. Type the details of the signoff, such as the reason, in the *Comments* field.

5. Click *OK*.

The signed-off failures of requirements of the type *Manual* are also documented in the verification reports.

## Setting Requirements for External References

This section details the following tasks.

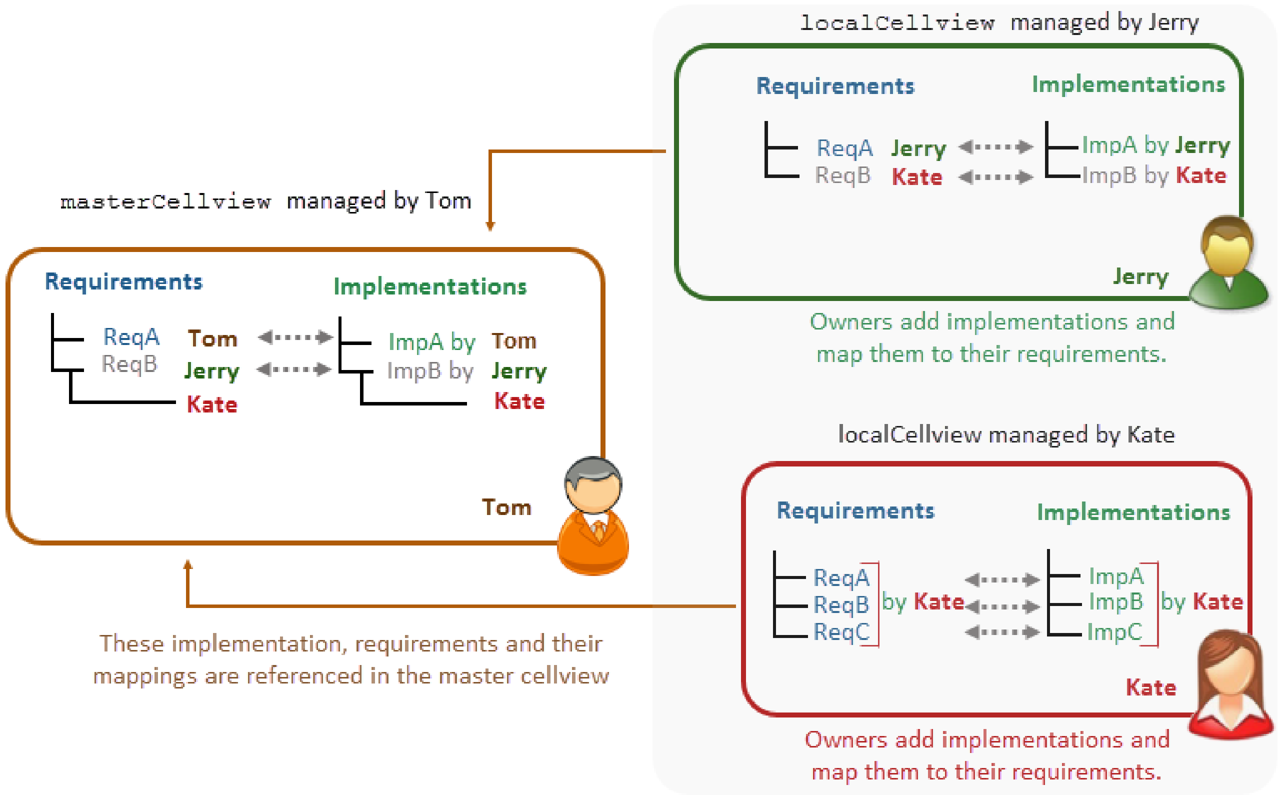
- Task 11: Understand How Verifier Supports Multiple External References
- Task 12: Set Owners for Requirements
- Task 13: View the Mapped Implementations from the Designer Cellview in the Master Cellview

### Task 11: Understand How Verifier Supports Multiple External References

As a verification project manager, you can set up an analog design verification project with multiple cellviews. You can then import existing Verifier cellviews as external references, and maintain your master cellview. Requirement designers can add implementations and map them with their requirements in their local cellviews. These implementations become available to the master cellview through reference. The verification project manager can view the latest verification status of requirements assigned to verification designers.

To be able to use Verifier in a multi-user setup, understand how the setup works.

The following figure illustrates how you can manage a design verification project with multiple requirement owners.


localCellview managed by Jerry
Requirements
Implementations
ReqA Jerry
ReqB Kate
ImpA by Jerry
ImpB by Kate
Jerry
Owners add implementations and map them to their requirements.
masterCellview managed by Tom
Requirements
Implementations
ReqA Tom
ReqB Jerry
Kate
ImpA by Tom
ImpB by Jerry
Kate
Tom
localCellview managed by Kate
Requirements
Implementations
ReqA
ReqB
ReqC
by Kate
ImpA
ImpB
ImpC
by Kate
Kate
Owners add implementations and map them to their requirements.
These implementation, requirements and their mappings are referenced in the master cellview

Assuming that you are the project manager, you typically follow these steps for managing a verification project with multiple designers:

1. Add requirements to the master Verifier cellview of your verification project.
2. Add owners of requirements.

   **Notes:**

   - ❑ The default owner of a new top-level requirement is the current Verifier user name. The default owner of a new sub-level requirement is the same as the owner of its parent requirement.
   - ❑ You can set the requirement–implementation mapping of a requirement if you are the owner of that requirement. You typically use Verifier as the assigned requirement owner for defining the mappings. If you want to use Verifier as another user, choose *Edit — Preferences*, and set the user name in the *Current User Name* field.

- ❑ If a requirement is without an owner and has been mapped to an implementation, you cannot specify its owner. In this case, you can delete the mapping and then specify the owner.
- ❑ Specifying an owner for a requirement has no impact on the verification status. The *Owner* name is specified only to assist in the identification of the designers currently working on a requirement.

3. Designers add implementations in their own cellview and map them with their requirements. Designers can also run simulations and monitor the verification status of the requirements they own.

   By default, designer cellviews display only the local requirements. Designers can add local requirements in their owner cellviews.

   **Note:** Only the owners of the failed requirements or the requirements of the type *Manual* can sign off those requirements from their owner cellviews. The project manager can review the sign off details.

4. You, as the verification project manager, can review and run the implementations added by designers from the master cellview.

   The master cellview includes the implementation information set in the designer cellviews through reference. You can monitor the verification status of all the requirements local to different designers.

**Task 12: Set Owners for Requirements**

Open the cellview `amsPLL/TOP_verification_multiuser/verification`. This is the master cellview of the designer cellviews.

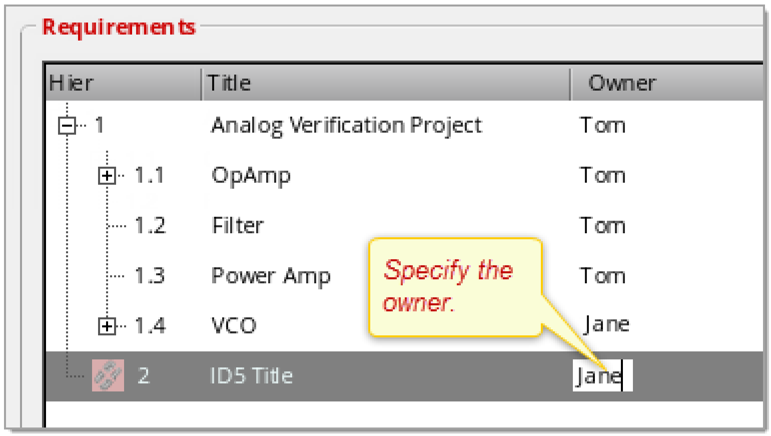
Ensure that the `Owner` column is displayed. If it is not displayed, right-click on the column headers in the Requirements pane and select *Owner*. All the requirements are assigned to the owner `Tom`.

Add a requirement `2`. Ensure that the owner of this requirement as `Jane`.

After you create the new requirement, the default owner of that requirement is `Tom`. This is because the current user of the master cellview is `Tom`.

### To set an owner for a requirement, do one of the following

- ➡ Select the requirement, click the *Owner* field, and type the name of the requirement owner.



Requirements

| Hier | Title | Owner |
|---|---|---|
| 1 | Analog Verification Project | Tom |
| 1.1 | OpAmp | Tom |
| 1.2 | Filter | Tom |
| 1.3 | Power Amp | Tom |
| 1.4 | VCO | Jane |
| 2 | ID5 Title | Jane |

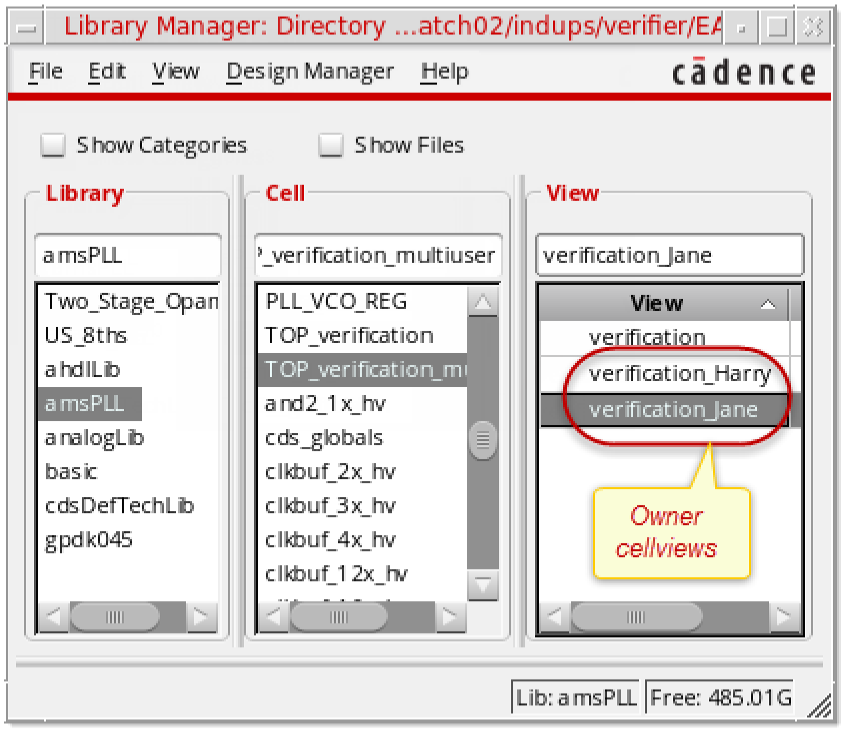
For this task, type `Jane` as the owner of the new requirement. Consider that the new requirement is `2`.

- ➡ Choose *Edit — Open Requirement Editor* and specify the owner in the requirement editor.
- ➡ Select requirements, right-click, choose *Set Owners for Selected Requirements*, type the owner, and click *OK*. Use this method to set the same owners for multiple requirements.

### To open an owner cellview from Library Manager

1. Choose *Tools — Library Manager* from Virtuoso CIW to launch Library Manager.
2. Select the library and cell.

**3.** In the list of views, identify the owner cellview. The default format is *`masterView_ownerName`*.



**4.** Double-click the owner cellview you want to open.

For this task, open the owner cellview of `Jane`.

The owner cellview has the requirement `2` that you added and assigned to `Jane`.

### Task 13: View the Mapped Implementations from the Designer Cellview in the Master Cellview

In the designer cellview of `Jane`, the following implementation cellview was added with the `Active` history.

```
amsPLL/PLL_VCO_320MHZ_tb/maestro
```

The requirements were mapped with the outputs in the designer cellview, as mentioned below.

- Map `1.4.1` to the output `freq_check`.
- Map `1.4.2` to the output `max_current`.
- Map `1.4.3` to the output `avg_current`.

Open the master cellview `amsPLL/TOP_verification_multiuser/verification` and check if the implementation and mapping are available.

#### To add an implementation cellview in the owner cellview

1. Click *Add Implementation* in the *Implementations* pane of the *Setup* tab.

   The selection form displays.
2. Select the implementation cellview.

   For the task, select `amsPLL/PLL_VCO_320MHZ_tb/maestro`.
3. Click *OK*.

   If the selected cellview has multiple histories, select the history.
4. Click *OK*.

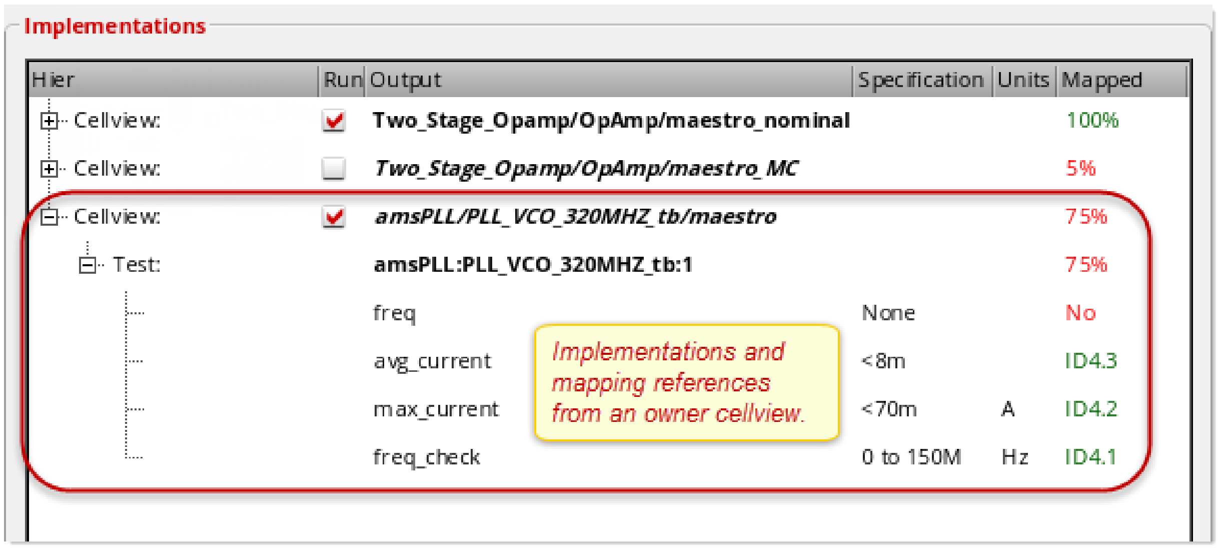
The implementation cellview is added.

#### To verify if the master cellview contains implementations and mappings set in owner cellviews:

1. Open the master cellview.

   For the task, open `amsPLL/TOP_verification_multiuser/verification`.

**2.** Check the presence of the implementations and mappings that were set in the owner cellviews.



Implementations

| Hier | Run | Output | Specification | Units | Mapped |
|---|---|---|---|---|---|
| Cellview: | ✓ | Two_Stage_Opamp/OpAmp/maestro_nominal | | | 100% |
| Cellview: | | ***Two_Stage_Opamp/OpAmp/maestro_MC*** | | | 5% |
| Cellview: | ✓ | ***amsPLL/PLL_VCO_320MHZ_tb/maestro*** | | | 75% |
| Test: | | amsPLL:PLL_VCO_320MHZ_tb:1 | | | 75% |
| | | freq | None | | No |
| | | avg_current | <8m | | ID4.3 |
| | | max_current | <70m | A | ID4.2 |
| | | freq_check | 0 to 150M | Hz | ID4.1 |

Implementations and mapping references from an owner cellview.

The referenced cellview name appears in bold italic text.

try

Open the master cellview. For any new requirements you add to this cellview, the default owner is `Tom`, the current user.

Change the current user. For this, click *Edit — Preferences* and type a different user name in the *Current User Name* field available in the *General Options* tab. You can use any name of your choice. When you click *OK*, Verifier identifies the requirements owned by `Tom` and prompts you to change the ownership of those requirements to the new user.

Click *No* for now so that the ownership is not changed. Then, add a new requirement. The owner of that requirement is the new user you specified.

## Setting Preferences

**Task 14: Review the Virtuoso ADE Verifier Preferences Form**

You can customize the behavior of Verifier by setting your preferences in the Virtuoso ADE Verifier Preferences form. In this task, review the options you can set as preferences.

**To set your preferences**

1. Choose *Edit — Preferences*.

   The Virtuoso ADE Verifier Preferences form displays.

   Review the options in each tab.

2. Set your preferences as required.
3. Click *OK*.

This tutorial includes useful information on setting preferences for various purposes, which are listed below. You can explore and use other preference options available through the Virtuoso ADE Verifier Preferences form.

- Set the specification search order.
- Specify the current user name.
- Retain mapping information when a requirement is renamed.
- Set Verifier to run multiple simulations in parallel.
- Specify a user-defined directory to store run summary data.
- Set Verifier to not resimulate an implementation when no changes were made.
- Set Verifier to display HTML reports in the default web browser.
- Set the batch run options.
- Set Verifier to use a specific job policy.
- Set custom fields for the project and its requirements.
- Set the time interval between implementation change checks.