

Synopsys[®] VC Formal Tutorial

Connectivity Checking Application (CC)

Version 1.0 | 16-May-2023



Portland State University

©2023

Disclaimer:

The contents of this document are confidential, privileged, and only for the information of the intended recipient and may not be published or redistributed.

The design example in this tutorial is not supplied. You need to use your own files to follow the tutorial steps and instructions

Table of Contents

Introduction	3
About and Usage of CC	4
Design Files	5
TCL File	6
Application Setup	7
Invoking VC Formal GUI	8
User Interface Details	8
Loading TCL File	9
Invoking VC Formal Along with TCL File	10
Running Files	11
Detecting Errors	12
Source Tracing	14
Debugging	15
Resolving Errors	20
Restarting VC Formal	25
Appendix	27
<i>Table 1.1. CC App Functions/commands</i>	28

Introduction

VC Formal uses TCL (Tool Command Language) scripts to tell it what to do. The script can define the app within VCFormal that we want to use, the files we are working on, how we are mapping them, the clock cycles, and more. Instead of diving into the details of TCL scripts, we will use templates. One would then simply need to modify those templates to interact with VC Formal as needed in their project.

To begin, you need to set up the VNCserver as shown in the previous tutorials and open the Linux GUI. For better practices and to keep everything organized, we create a main folder for the design we want to analyze, and in this case, I called mine “CC_Example”. Don’t use spaces when naming the files and folders.

Inside that folder, we create two folders: one is Design, where you put the actual Verilog or SystemVerilog designs we want to analyze, and the other is Run, where we put the TCL that will run the VCFormal CC analysis for us.

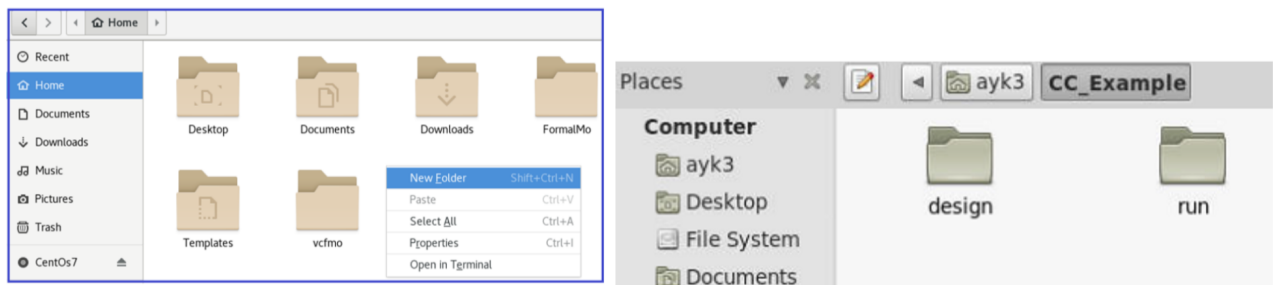
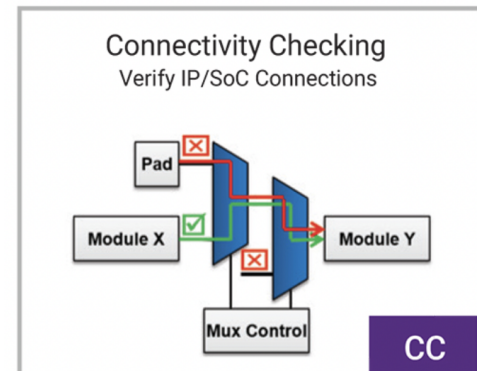


Figure 1. Creating folders to organize design and TCL files.

About and Usage of CC

The “CC” app in VC Formal is used as a connectivity verification tool to prove conditional wiring. CC app is a powerful debugging tool that includes automatic root-cause analysis of unconnected connectivity checks which saves significant debug time. It expedites SoC connectivity verification at the top level along with the connection between IP blocks.

The CC application takes the design (RTL) and the connection specifications as inputs, and verifies if the connection specifications are good or not. CC app is used to verify the correctness of a design's connectivity by analyzing the circuit's logic and ensuring that signals are properly connected between modules.



Design Files

In the Design folder, you'll put in the SystemVerilog design file. The TCL file will be put in the Run folder. We suggest you name your folders with lowercase letters, as VC Formal is case-sensitive.

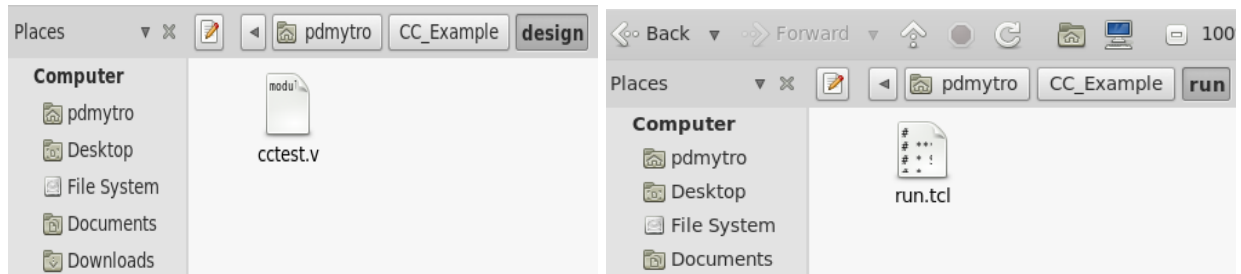


Figure 2. Showing the SystemVerilog and TCL files in the correct folder locations.

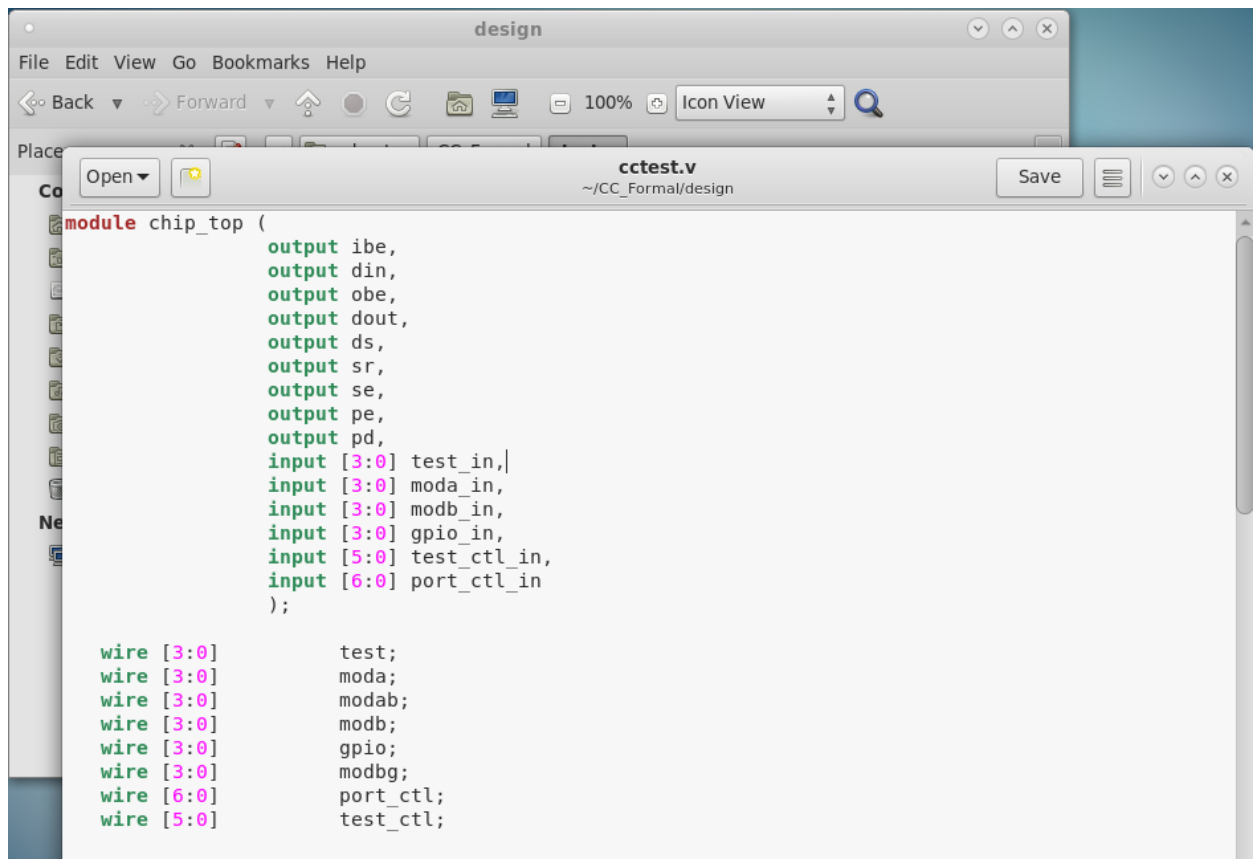


Figure 3. Example of the design we are using in this tutorial.

In order for VC Formal to map your design, you need to acknowledge and keep note of your exact module name, as highlighted in *Figure 3* above. This will come in handy when you create your TCL file. To make things simple, we suggest naming your design file after its corresponding module name (more on this in the TCL File section below).

TCL File

Next, we will set up the TCL file. Below is the TCL file (*Figure 4*), which you can use as a template for your functional checks on VC Formal.

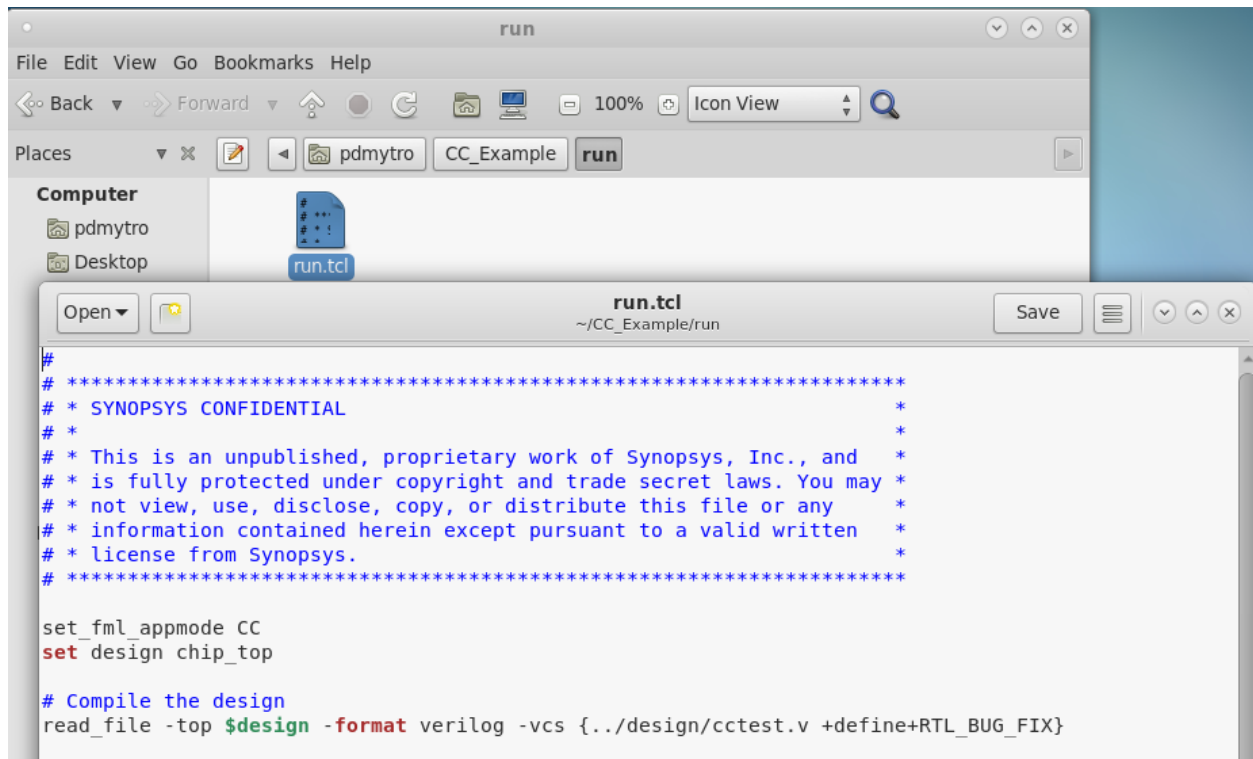


Figure 4. Annotated TCL template file.

- (1) Instruction that sets the appmode to CC in VC Formal.
- (2) Name of the main module as established in the Design file.
- (3) The property type identifier switch name for desired CC analysis.
- (4) Design file location so VC Formal knows where to find the file.

As mentioned before, VC Formal is case sensitive, and in order for it to map your designs, you will need to use the same module name in the TCL script **(2)** and the module name in the design file. For example, we can see that our module name in the TCL script is the same as the module name in the design file in *Figure 3*.

The file name (train_Controller.sv) can be named however you want.

Application Setup

There are multiple ways to invoke the VC Formal GUI and load the TCL script.

In this tutorial, we will show two methods for doing so. When using either method, it is important that you open the terminal within the appropriate “Run” folder associated with the AEP app.

- [Invoke VC Formal GUI](#), then manually load the TCL script in the application:

```
$vcf -gui
```

OR

- [Invoke VC Formal GUI and TCL script](#) in one command:

```
$vcf -f run.tcl -gui      or      $vcf -f run.tcl -verdi
```

‘run.tcl’ is the name of the TCL file we are using. If your file name differs from this, you will need to change it accordingly in this command.

The “-gui” switch opens VC Formal in the GUI, and it’s equivalent to the switch “-verdi”.

We will go through both of these methods in the following sections.

Invoking VC Formal GUI

To proceed with invoking VC Formal:

- 1) Inside the “run” folder, right-click the whitespace and choose “Open in Terminal”
- 2) In the terminal, type in the command:

```
vcf -gui
```

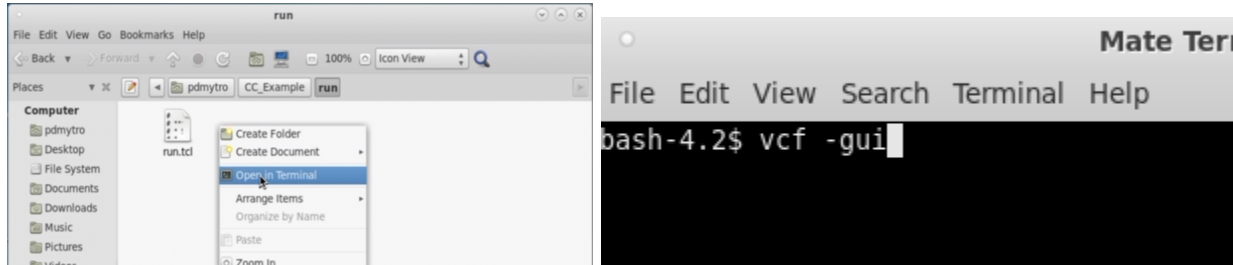



Figure 5. Invoking VC Formal in the terminal.

Note: You may have to wait a few seconds for the program to start up.

You should then see the VC Formal GUI as shown in *Figure 6* below. You can toggle between showing Targets, Constraints, or both Targets and Constraints window by clicking the blue box icon in the upper right-hand of the application **(1)**.

It may look like any of these three icons: 

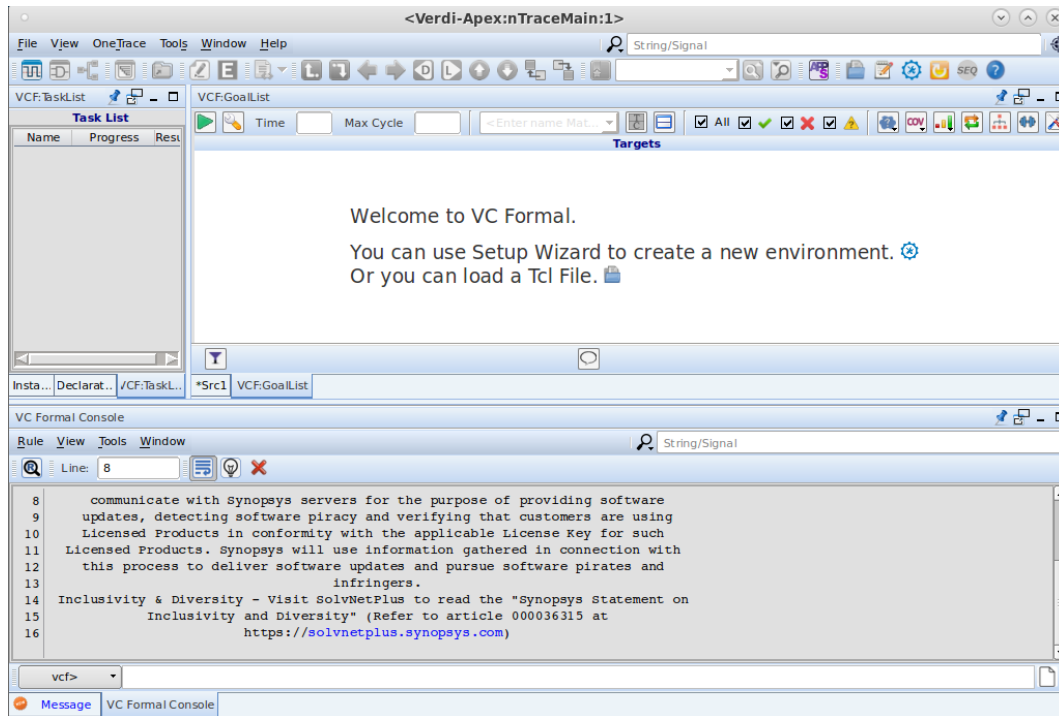



Figure 6. VC Formal GUI introductory screen.

Then load a TCL script by clicking on the  icon **(2)** as shown in Figure 6.

Next, select the “run.tcl” file we have in the “run” folder:

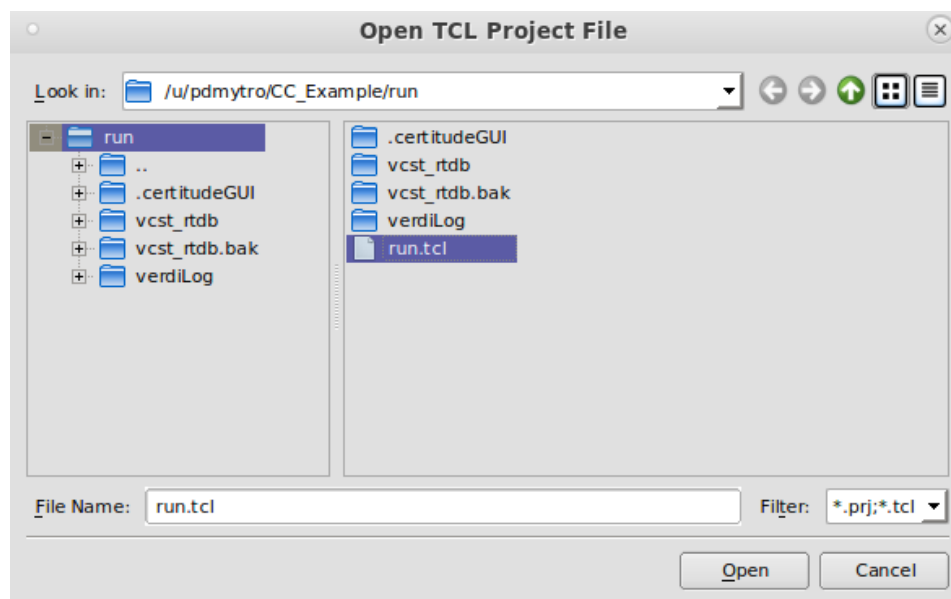


Figure 7. Selecting TCL file.

Invoking VC Formal Along with TCL File:

To proceed with invoking VC Formal:

- 1) Inside the “run” folder, right-click the whitespace and choose “Open in Terminal”
- 2) In the terminal, type in the command:

```
vcf -f run.tcl -gui
```

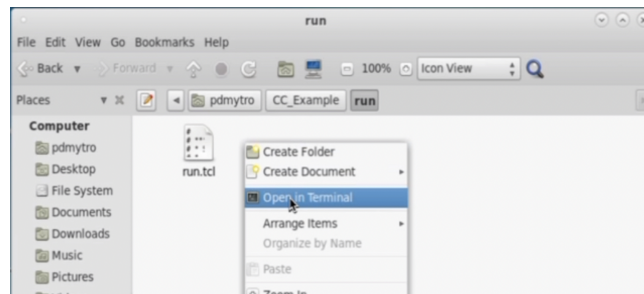


Figure 8. Opening terminal in the ‘run’ folder.

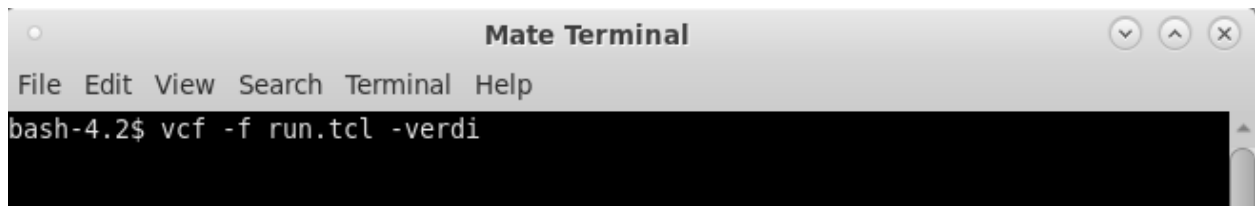


Figure 9. Invoking VC Formal and TCL script in the terminal.

And that's it!

Running Files

After successfully invoking VC Formal GUI and loading the TCL script in the above methods, your screen should have contents in the *VCF:GoalList* tab and look something like this:

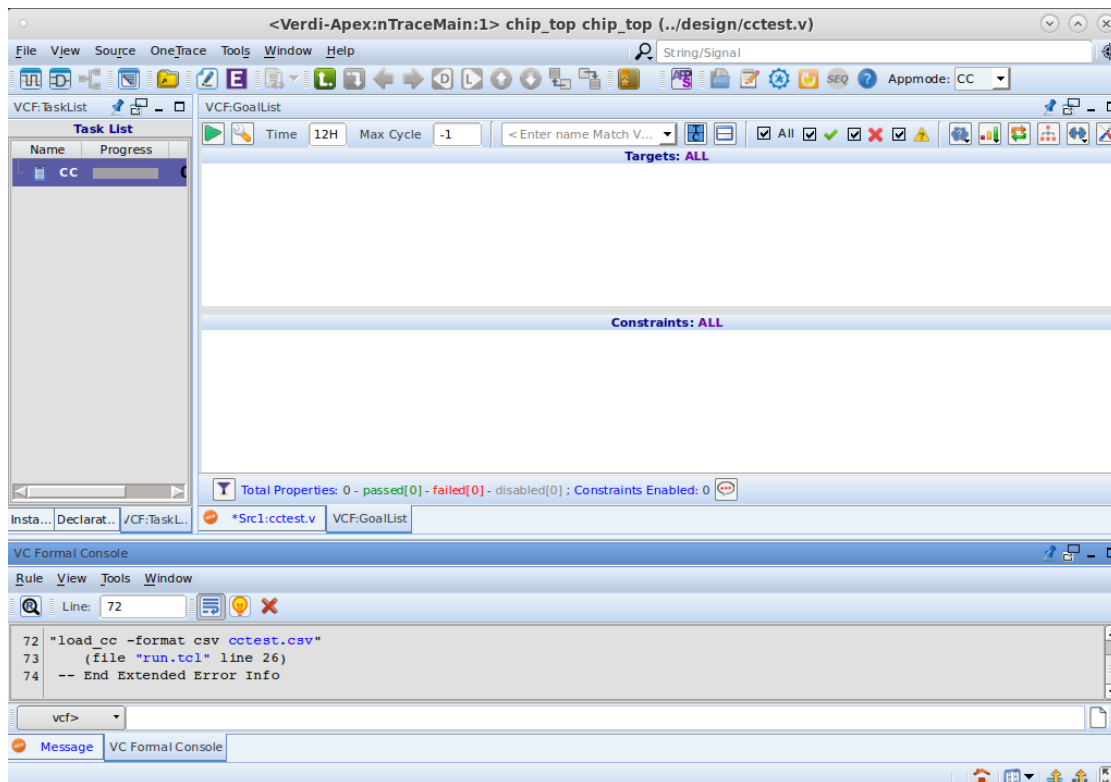



Figure 10. Screen after loading TCL script.

Now, go ahead and run the verification analysis by clicking on the play  icon in the upper left corner of the *VCF:GoalList* tab.

Detecting Errors

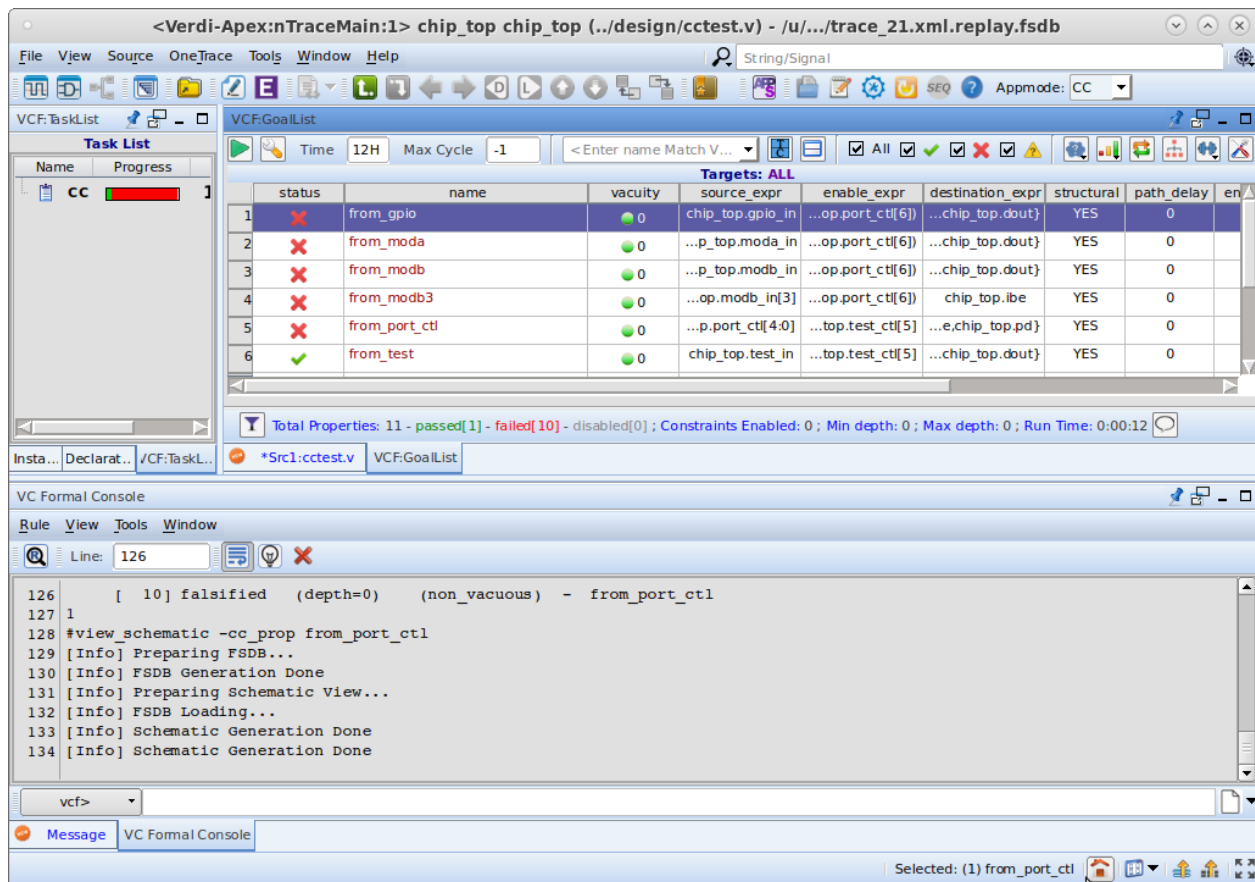





Figure 11. Screen after running TCL script and the status given = ✗.

In Figure 11 above, we see one  icon, this shows that the connection on that line has been proven and passes.

We also see ten  icons; indicating that the connection on these lines cannot be proven.

The  mean that the application is detecting an error in the connection from the source to the destination.

On the left under *Task List*, we can see that VC Formal was given one task by the CC app. You can hover over the numbers under *Result* to see the results in detail. You may need to alter the tab size by dragging the side panels or scrolling to the left:

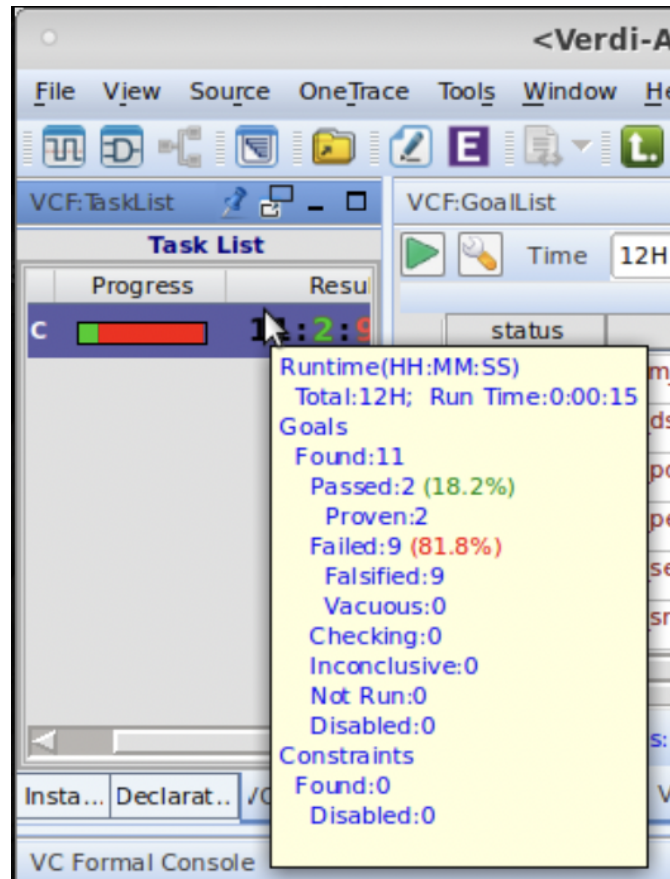




Figure 12. Results of the analyzed script.

Debugging Failures

Source tracing is a great way to determine where exactly our errors lie and to get a deeper look into what the issue is. To start, go ahead and double-click on an  icon.

We are going to right-click on the fifth  (line 5) from *Figure 11*. Select New Schematic Path.

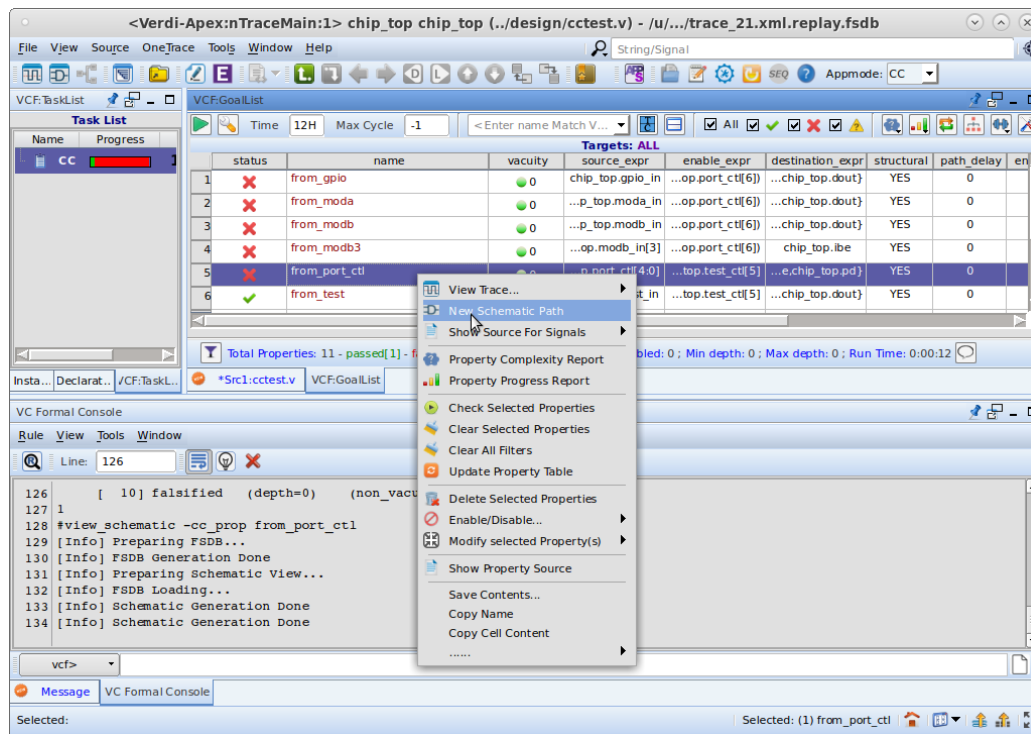


Figure 13. Examining the failed connectivity check in our design.

You should then see a generated schematic as shown below from figure 13:

On the left in *Figure 13* above, we see *Support-Signals* in green text. Go down three lines to the text that says *counter[2:0]* and right-click on it.

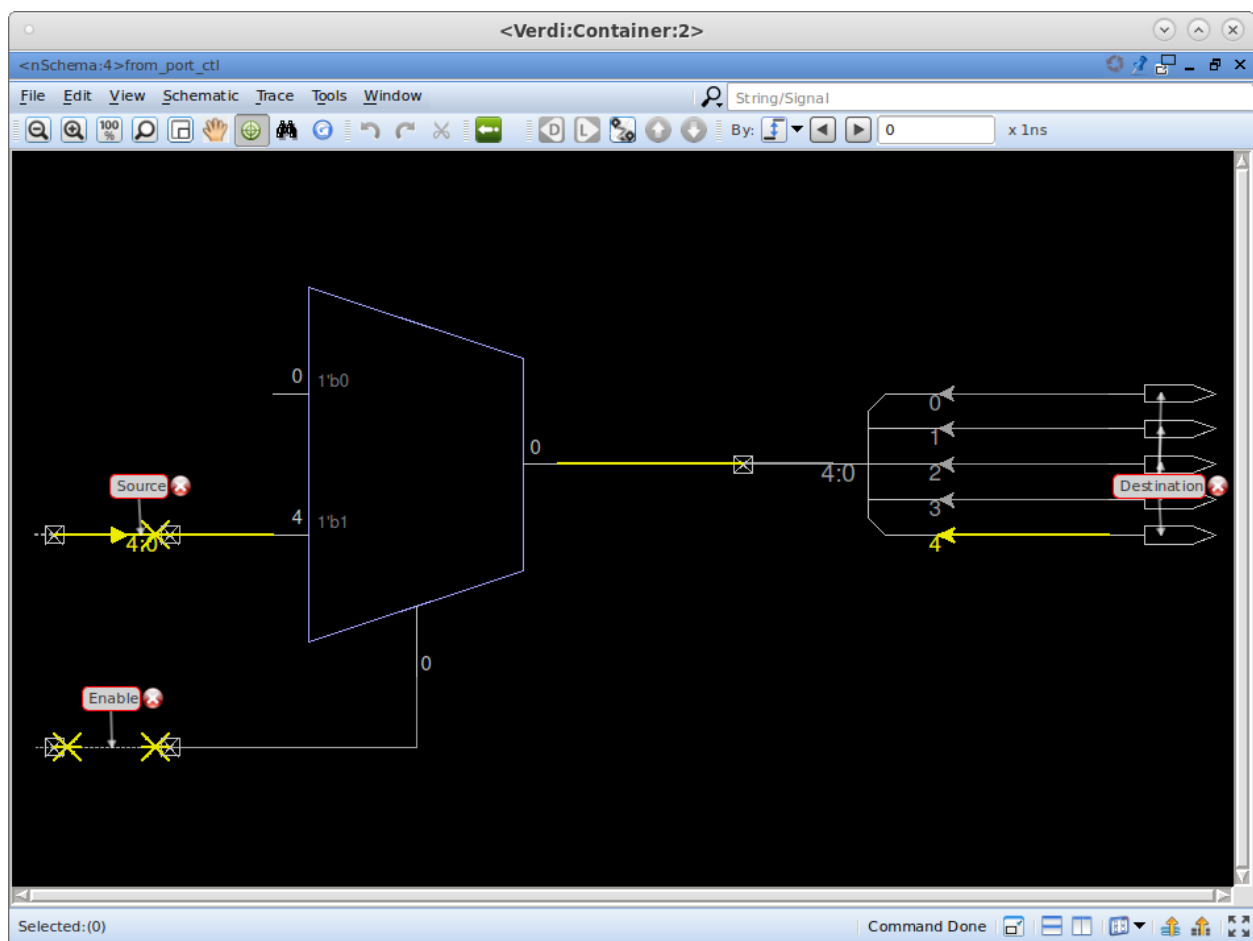


Figure 14. Examining the schematic

As shown in *Figure 14* above, this is the general method of source tracing:

Right-click the signal → Show OnceTrace Signals → Driver

You can also use the short-cut keys: **ALT + SHIFT + D**

By tracing the source, we are essentially backtracking it to the driving signal of the output in order to find the discrepancy. We then get these additional waveforms, showing the driving signal of this “counter” signal, which is the previous counter value.

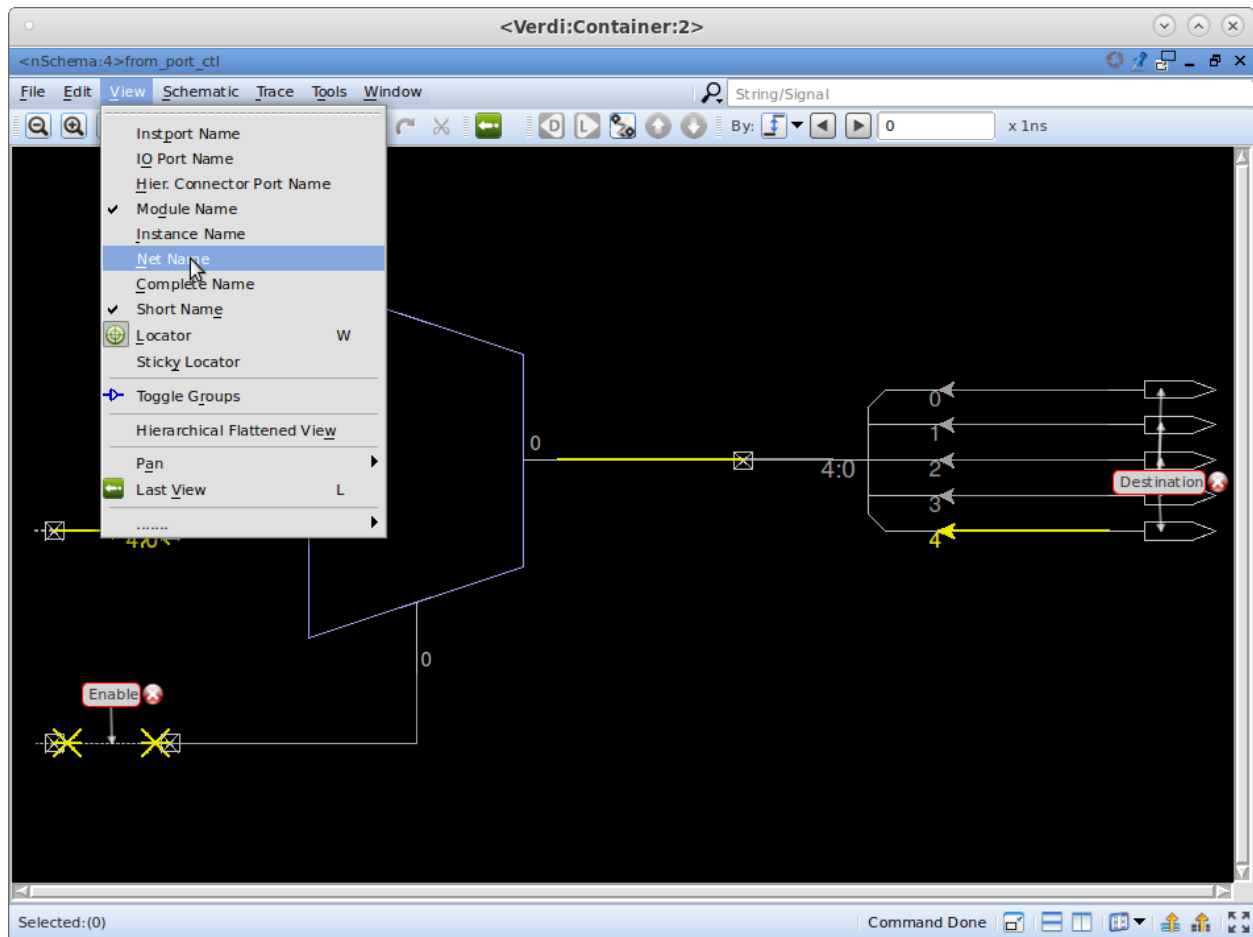


Figure 15. Showing schematic with how to enable net name view

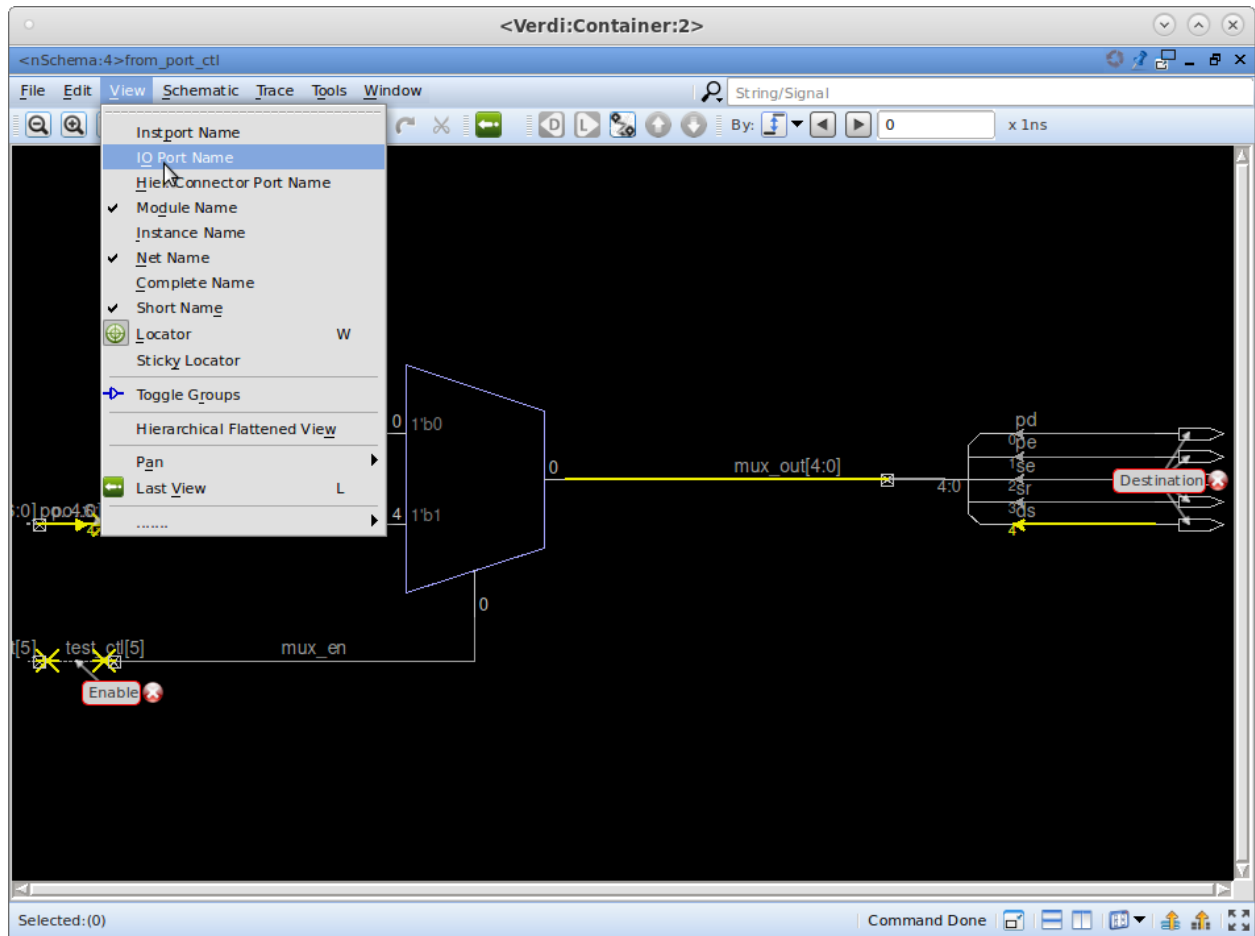


Figure 16. Showing schematic with how to enable IO Port name view

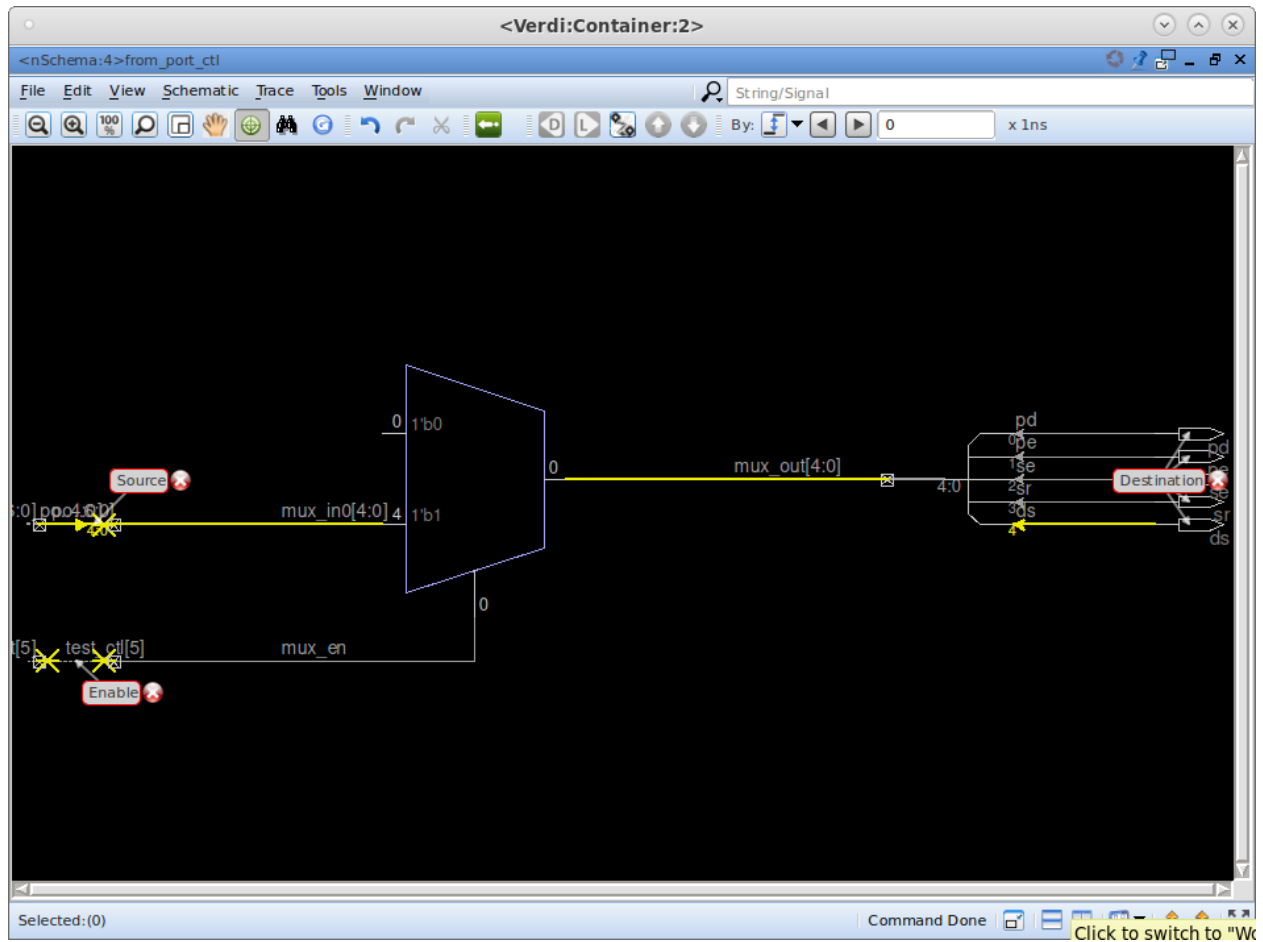


Figure 17. Net name and IO port name enabled helping to identify source of error

Resolving Errors

Upon debugging the failure on line 5 the schematic shows that the source does not equal to the destination therefore causing an error.

5	✗	from_port_ctl	0	...p.port_ctl[4:0]	...top.test_ctl[5]	...e,chip_top,pd}	YES	0
---	---	---------------	---	--------------------	--------------------	-------------------	-----	---

Figure 18. Identified Error on line 5

The schematic showing the cause of the error. The source is chip_top.port_ctl[4:0], the destination is {ds,sr,se,pe,pd}, and the enable condition is ~chip_top.test_ctl[5]. In the schematic it can be seen that the mux enable signal is zero, therefore the source is not equal to the destination causing the error.

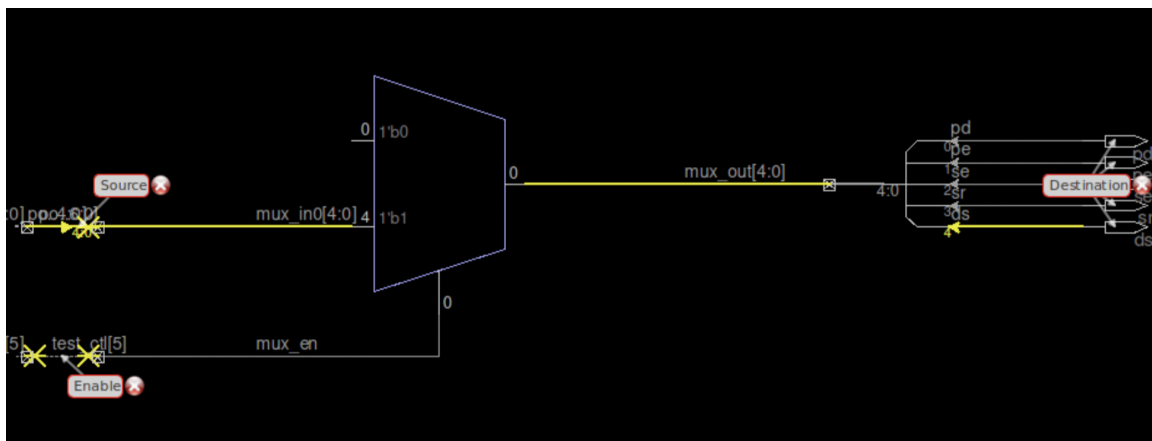


Figure 19. Identified errors in schematic

To fix this issue, and make the connection pass we need to go alter our design file and make the following changes:

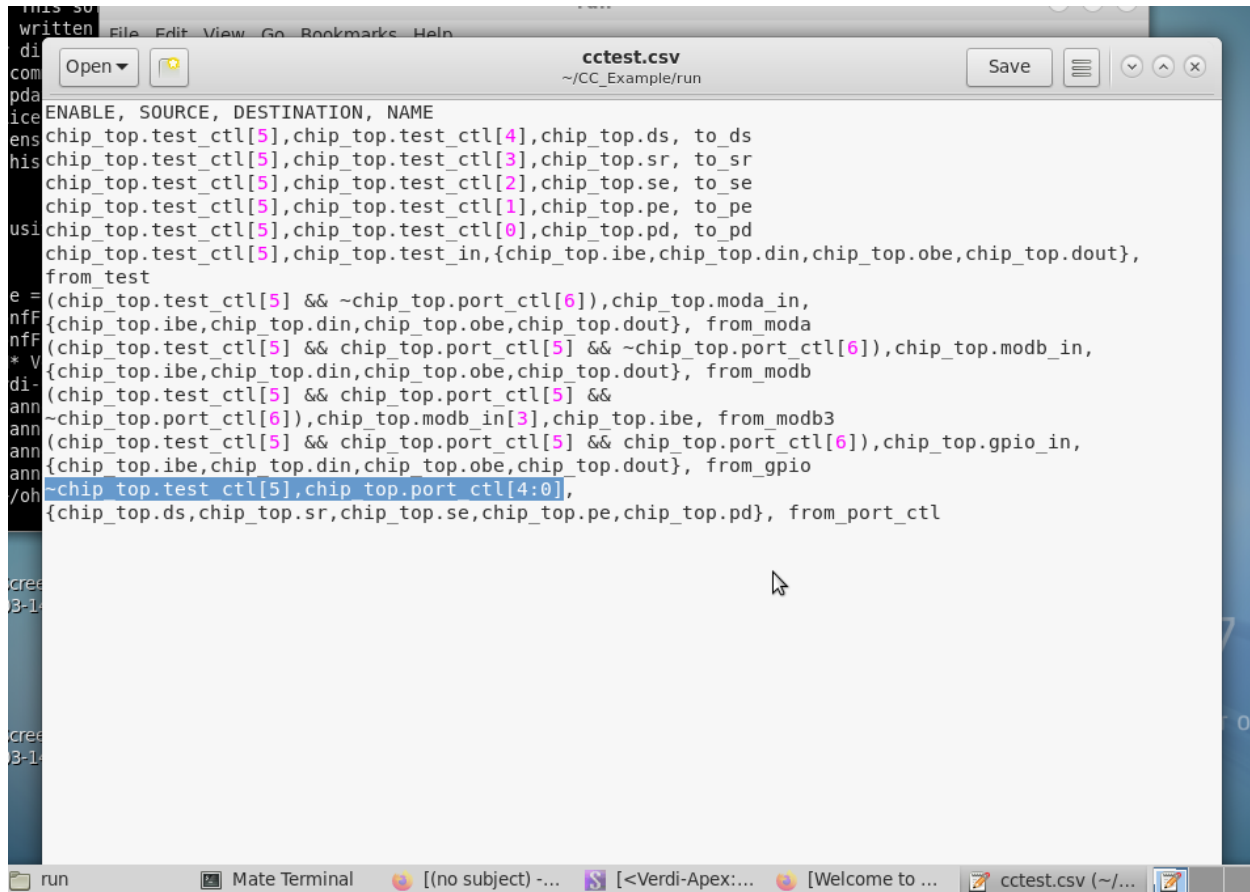


Figure 20. Alter design file to fix the error on the line highlighted in blue.

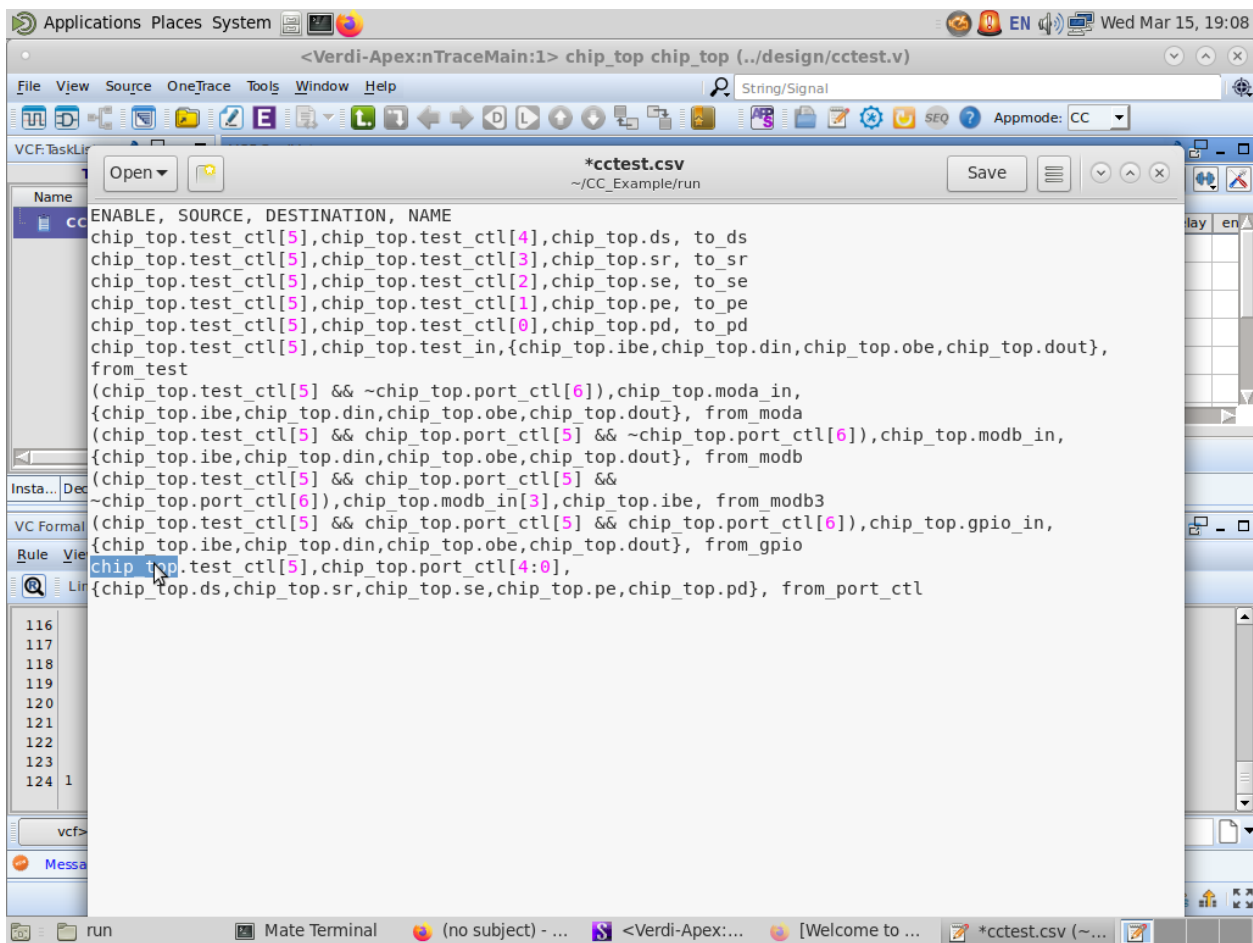


Figure 21. Remove the “~” from the last line and save the file.

Now return to the VC Formal Application.

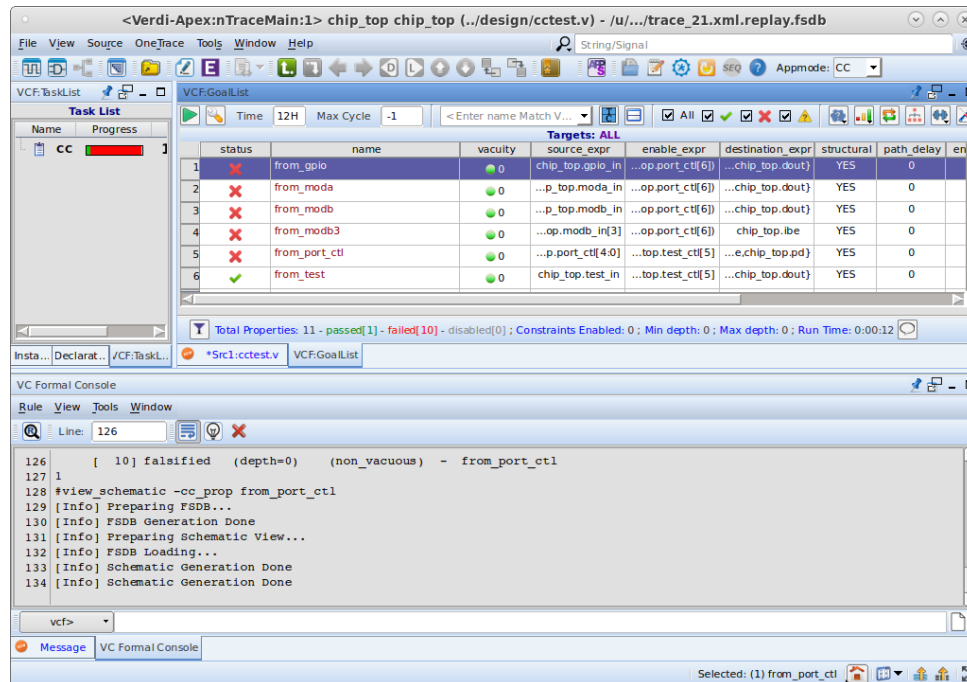



Figure 22. Restart VC Formal with the updated file by clicking the yellow arrow button in the top right corner next to “Appmode:” 

The connection now passes.

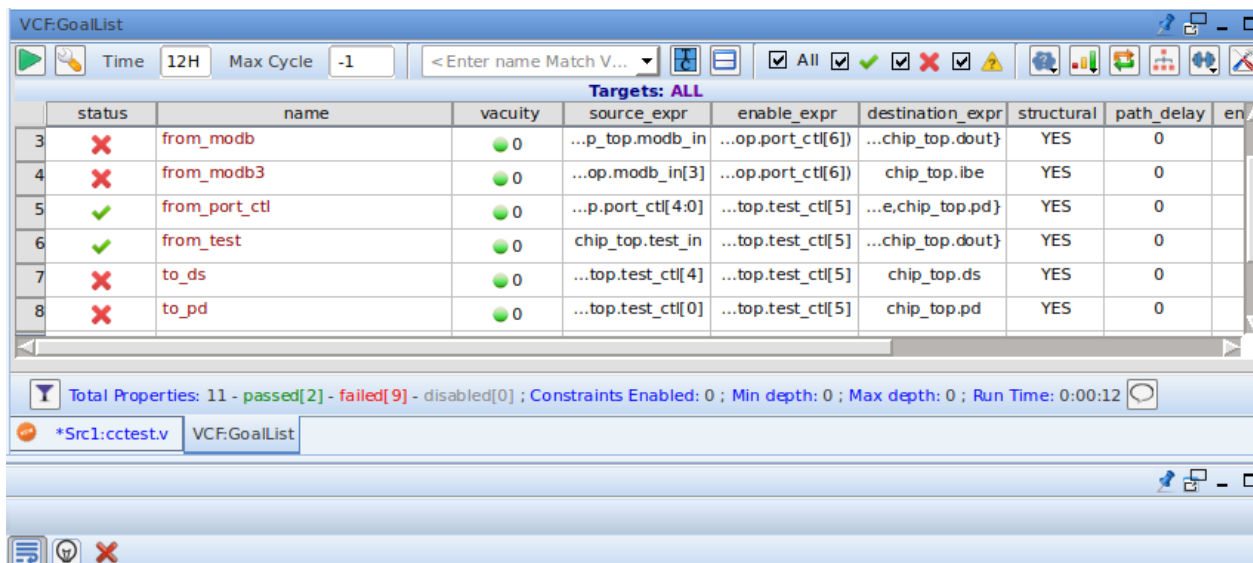


Figure 23. Error on line 5 resolved.

1	✗	from_gpio	0	chip_top gpio_in	...op.port_ctl[6]	...chip_top.dout	YES	0
2	✗	from_moda	0	...p_top.moda_in	...op.port_ctl[6]	...chip_top.dout	YES	0
3	✗	from_modb	0	...p_top.modb_in	...op.port_ctl[6]	...chip_top.dout	YES	0
4	✗	from_modb3	0	...op.modb_in[3]	...op.port_ctl[6]	chip_top.ibe	YES	0

Figure 24. Reviewing remaining errors.

The errors that remain should be reviewed in the same way that we reviewed the error on line 5. After the condition that is causing the connection to fail is identified on the remaining lines, resolve these errors in the design file. Save changes.

Next, to complete our changes, follow the steps below to restart VC Formal.

Restarting VC Formal


We can restart VC Formal by clicking on . If you invoked VC Formal along with the TCL script in the one-line command, then VC Formal will automatically load the same TCL file again. If you invoked VC Formal without the TCL script, then you will have to manually load the script again.



Figure 25. Location of the restart button in VC Formal window.

After the application and TCL file is loaded, click the green play button and we should see no errors.

The screenshot displays the Verdi-Apex: nTraceMain:1 chip_top chip_top (../design/cctest.v) interface. The VCF TaskList shows a single task 'CC' with a progress bar. The VCF GoalList table lists 6 goals, all with a status of '✓' and a vacuity of 0. The VCF Formal Console shows a list of 10 proven goals, all with a status of 'proven' and a vacuity of 0.

status	name	vacuity	source_expr	enable_expr	destination_expr	structural	path_delay	en
✓	from_gpio	0	chip_top_gpio_in	...op.port_ctl[6]	...chip_top.dout	YES	0	
✓	from_moda	0	...p_top.moda_in	...op.port_ctl[5]	...chip_top.dout	YES	0	
✓	from_modb	0	...p_top.modb_in	...op.port_ctl[6]	...chip_top.dout	YES	0	
✓	from_modb3	0	...op.modb_in[3]	...op.port_ctl[6]	chip_top.ibe	YES	0	
✓	from_port_ctl	0	...p.port_ctl[4:0]	...top.test_ctl[5]	...e.chip_top.pd	YES	0	
✓	from_test	0	chip_top.test_in	...top.test_ctl[5]	...chip_top.dout	YES	0	

Total Properties: 11 - passed[11] - failed[0] - disabled[0] ; Constraints Enabled: 0 ; Run Time: 0:00:12

VC Formal Console

Rule View Tools Window

Line: 113

```

113 [ 3] proven (non_vacuous) - to_pe
114 [ 4] proven (non_vacuous) - to_pd
115 [ 5] proven (non_vacuous) - from_test
116 [ 6] proven (non_vacuous) - from_moda
117 [ 7] proven (non_vacuous) - from_modb
118 [ 8] proven (non_vacuous) - from_modb3
119 [ 9] proven (non_vacuous) - from_gpio
120 [10] proven (non_vacuous) - from_port_ctl
121 1

```

vcf>

Message VCF Formal Console

Line: 1, Col: 8; Top

Figure 26. Results after fixing errors and restarting VC Formal and reloading TCL script.

All of the connections now pass.

Appendix

Function	Code	Description
Set App mode	set_fml_appmode CC	VC Formal App mode command for CC
Format	-format <format>	The format of the file to be loaded. The two formats are csv or table. The format option is optional, and default is csv.
Results Summary	-quiet	Option to hide the progress message and summary results.
CSV File Path	<filename>	Path of the properly formatted csv file to read.
Non-applicable Properties to CC	-not_applicable	Shows properties for which structural check is not applicable (disabled/constant src).
Disconnected Properties	-disconnected	Shows structurally disconnected properties only.
Report of Warnings	-warning	Specify this option to get a report of warnings. You can view the warnings file-wise. Appending the -warning switch with -list and -verbose, displays all warnings from the list and verbose.
Detailed Report Information	-verbose	Displays the summary of results and then provides detailed information about the report such as the status, connection name, line number and message of each property file-wise.
enable_delay	<enable_delay>	Enable time relative to source, that is, how long before src does enable need to be asserted (default 0).
offset_delay	<offset_delay>	Number of cycles after reset to start connectivity check.
clock	<clock-expr>	Optional clocking expression for the CC. The default is to check on the posedge of the clock declared with create_clock.
dest	<destination-signals>	Provide an ordered list of signals to use as destination of a connection to check. Wild card characters are supported

Table 1.1. CC App important functions and commands