

Synopsys VC Formal Tutorial

Sequential Equivalence Checking App (SEQ)



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contents of this document are confidential, privileged and only for the information of the intended recipient and may not be published or redistributed.

Equivalence Checking in Synopsys VC Formal SEQ App Youtube Tutorial

Link: <https://youtu.be/Dlci5naLkaY>

The video is unlisted, and is not to be shared outside of this class.

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
Design Files	4
TCL File	5
Invoking VS Formal	6
Loading TCL Script	8
Detecting Errors	9
Resolving Errors	13

Introduction

The “SEQ” app in VC Formal is used to verify the equivalence of two designs using formal methods. One is called the specification, and the other is the implementation. So, we need designs, or files (usually written in an HDL language like Verilog, SystemVerilog, VHDL, etc.).

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 “SEQ_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 SEQ analysis for us.

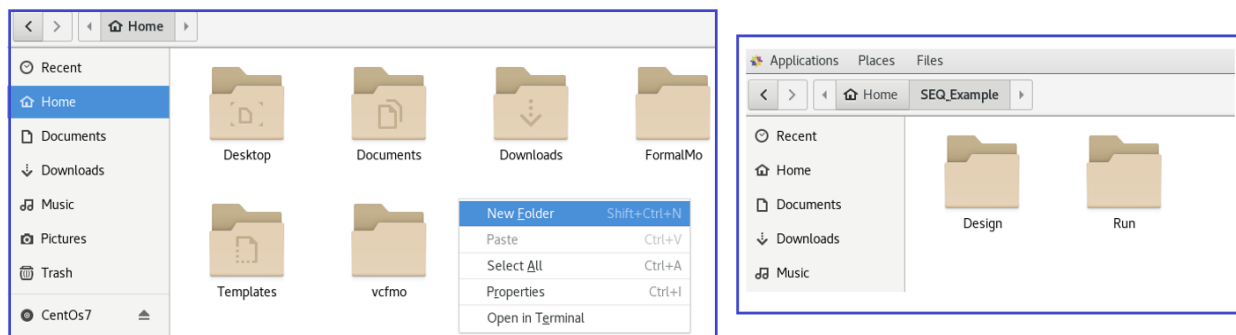


Figure 1. Creating folders to organize design and TCL files.

Design Files

In the Design folder, you'll put the two SystemVerilog design files. We are going to tell VC Formal to automatically map the two designs by name, so it's crucial to have the same input and output names in both the designs (don't use indices for example). The module names can be different, however.

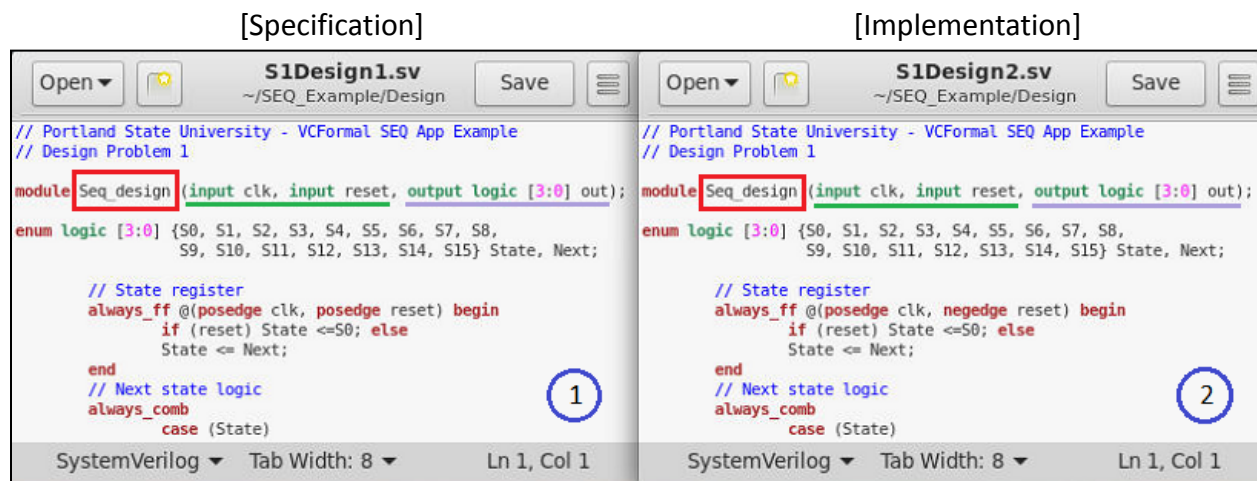


Figure 2. Showing the two example designs with same input and output module names.

- (1) S1Design1.sv (Specification file)
- (2) S1Design2.sv (Implementation file)

The two designs in Figure 2 above are identical, besides the negative edge triggered reset in the **always_ff** block in the implementation design (2). Later, you will see the effects this will have in our simulation.

TCL File

Next, we will set up the TCL file. Below is the TCL file (*Figure 3*), which you can use as a template for your equivalence checking on VC Formal.

```
run.tcl
~/SEQ_Example/Run

# Portland State University VC_Formal_Team_SEQ_App Tutorial

set_fml_appmode SEQ

# First, we compile the two designs (specification and implementation)
analyze -format sverilog -library spec ../Design/S1Design1.sv
analyze -format sverilog -library impl ../Design/S1Design2.sv
elaborate_seq -spectop Seq_design -impltop Seq_design

# Mapping the two designs by names (inputs, outputs and blackboxes)
map_by_name

## Creating clock and reset signals
create_clock -period 100 spec.clk
create_reset spec.rst -sense low

## Running a reset simulation
sim_run -stable
sim_save_reset

#mapping uninitialized registers using SEQ config
seq_config -map_uninit -map_x zero

# Running check command
#check_fv
```

Figure 3. Annotated TCL file.

- (1) spec is the specification file
- (2) impl is the implementation file
- (3) sverilog means that the designs are written in SystemVerilog

set_fml_appmode SEQ – this line is an instruction that sets the appmode to SEQ in VC Formal.

analyze -format sverilog -library

The two dots, “../” before the file location tells VC Formal to go to the main folder, then open the folder “Design” and look for the files “S1Design1.sv” and “S1Design2.sv”.

When you are running your own designs, you would need to change “S1Design1.sv” and “S1Design2.sv” to your file names.

elaborate_seq -spectop Seq_design -impltop Seq_design

This shows that the module name in both the specification and implementation designs are both named “Seq_design”. You will need to change this when running your own designs to its appropriate module names.

You can explore the other options in the TCL script if you like, but most importantly, when running your own designs, you need to change the file names in lines **(1)** and **(2)** from *Figure 3* and the module names in the **elaborate_seq -spectop** line. The module names can be the same, but they don’t have to.

Invoking VC Formal

Now, we can 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**

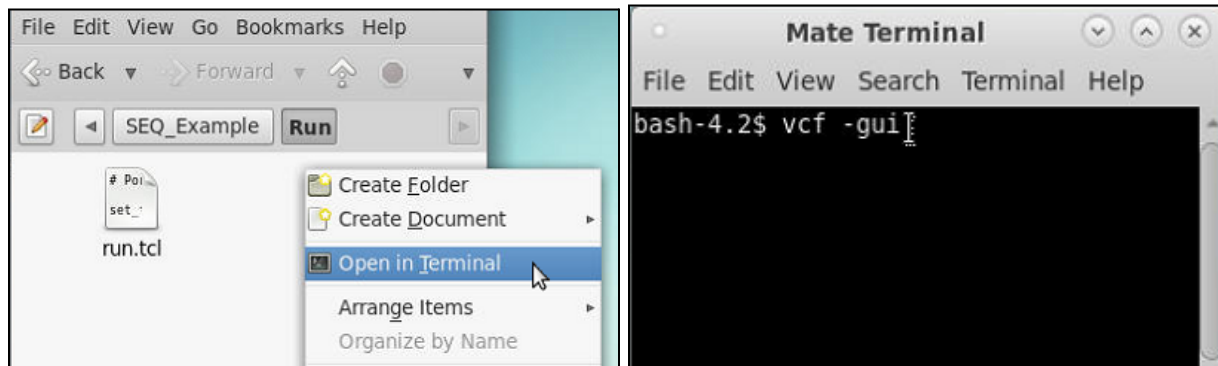



Figure 4. Invoking VC Formal in the terminal.

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 5* 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: 

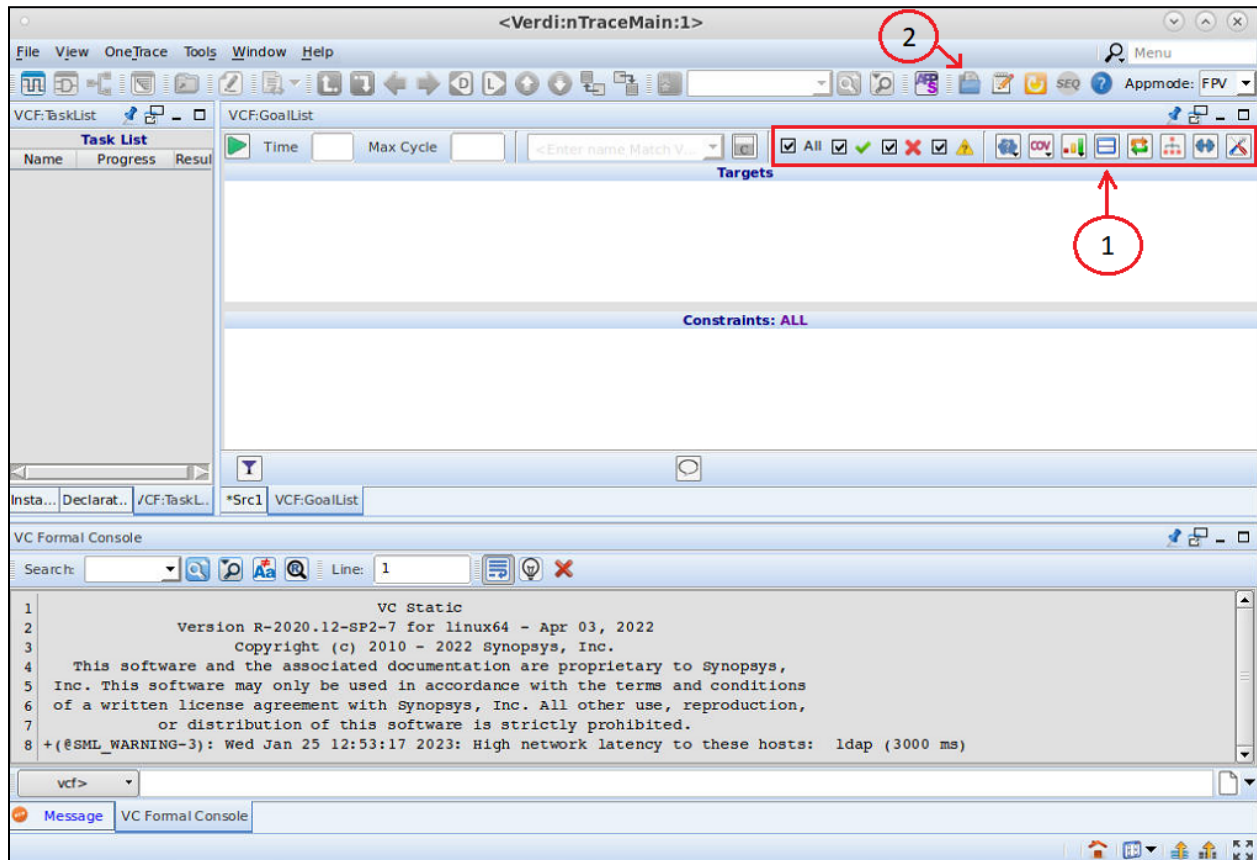


Figure 5. VC Formal GUI introductory screen.

Loading TCL Script

To load a TCL script, click on the  icon **(2)** as shown in *Figure 5*.

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

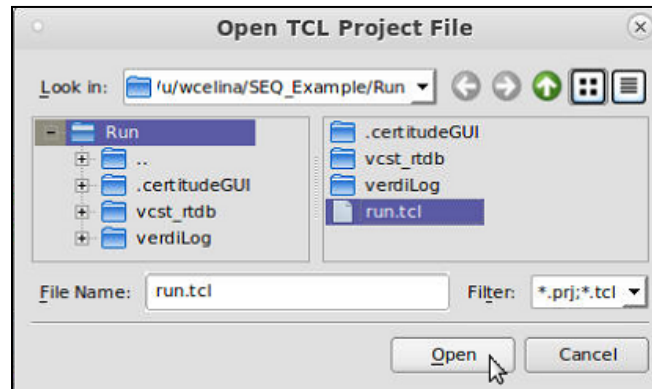


Figure 6. Selecting TCL file.

Your screen should look like this after clicking “Open”:

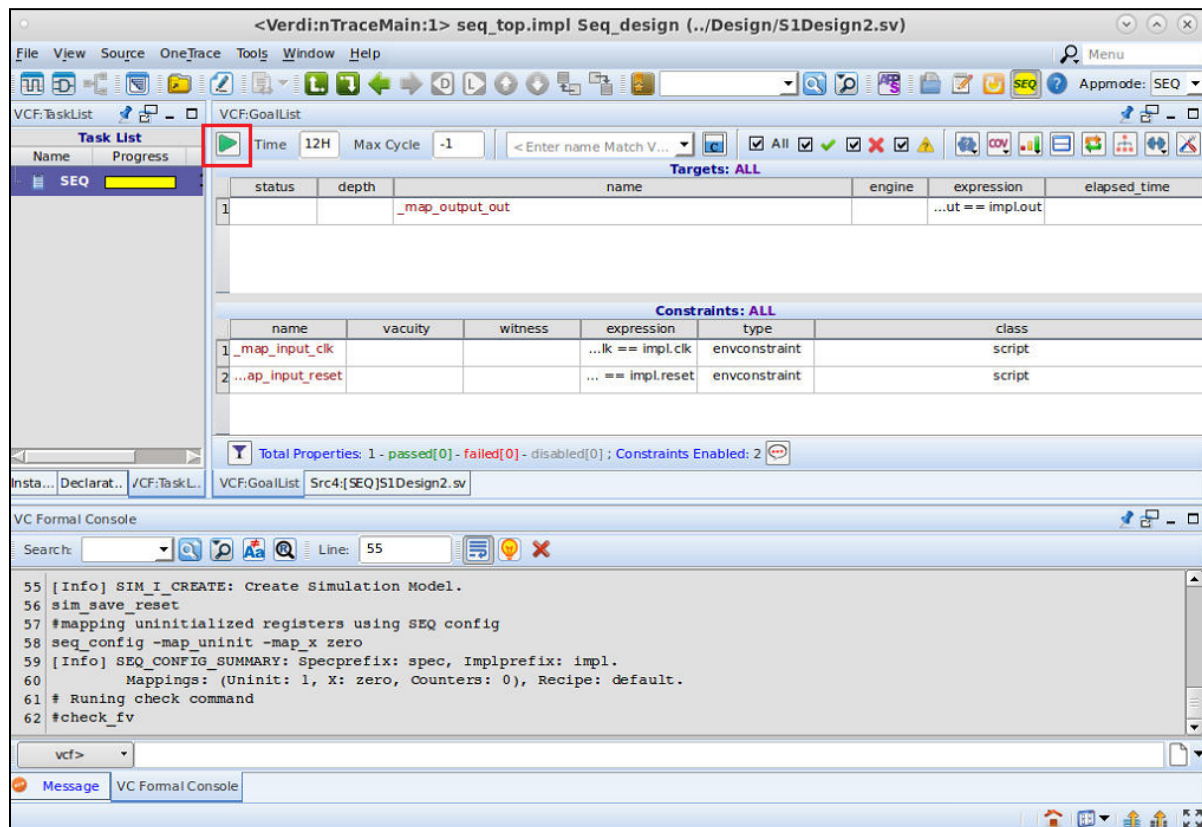




Figure 7. Screen after loading TCL script.

To run the analysis, click on the play  icon in the upper left.

Once the verification analysis is finished, we check the status icon. Here, we see the  icon under “status”, which means that the two designs are not equivalent.

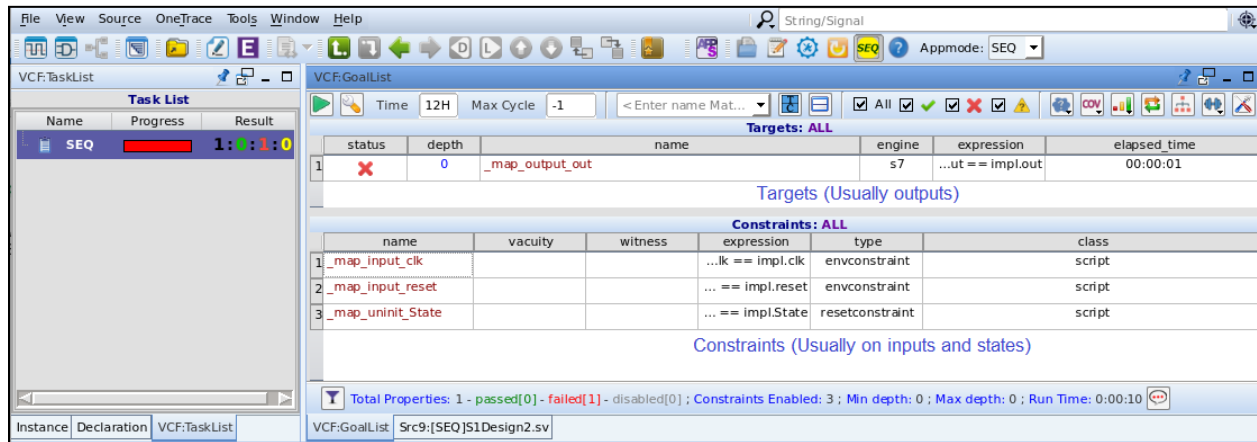


Figure 8. After running the script. Status given = ✖.

Detecting Errors

In the Task List on the left, hover over the results to see that VC Formal was given 1 task (one target, an output) to verify its equivalence, and it was shown that the two designs were not equivalent: “Failed/Falsified”. The other two outcomes would have been “Passed/Proved” and “Inconclusive” along with the additional options shown below.

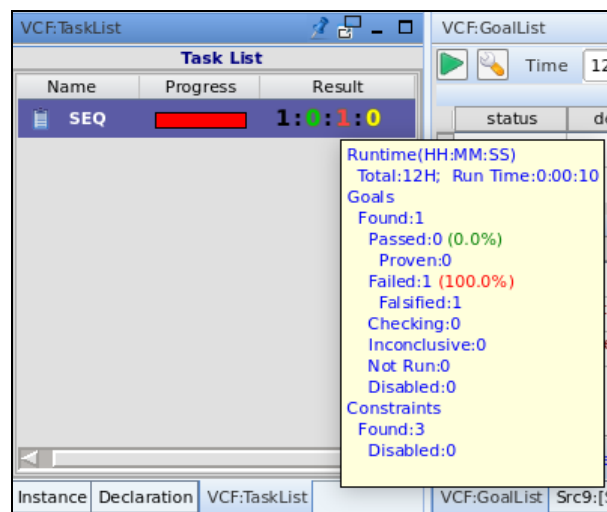



Figure 9. A closer look into the details of the analysis.

To check for the inequivalent part of the design, click on the  icon. We should then get the screen below:

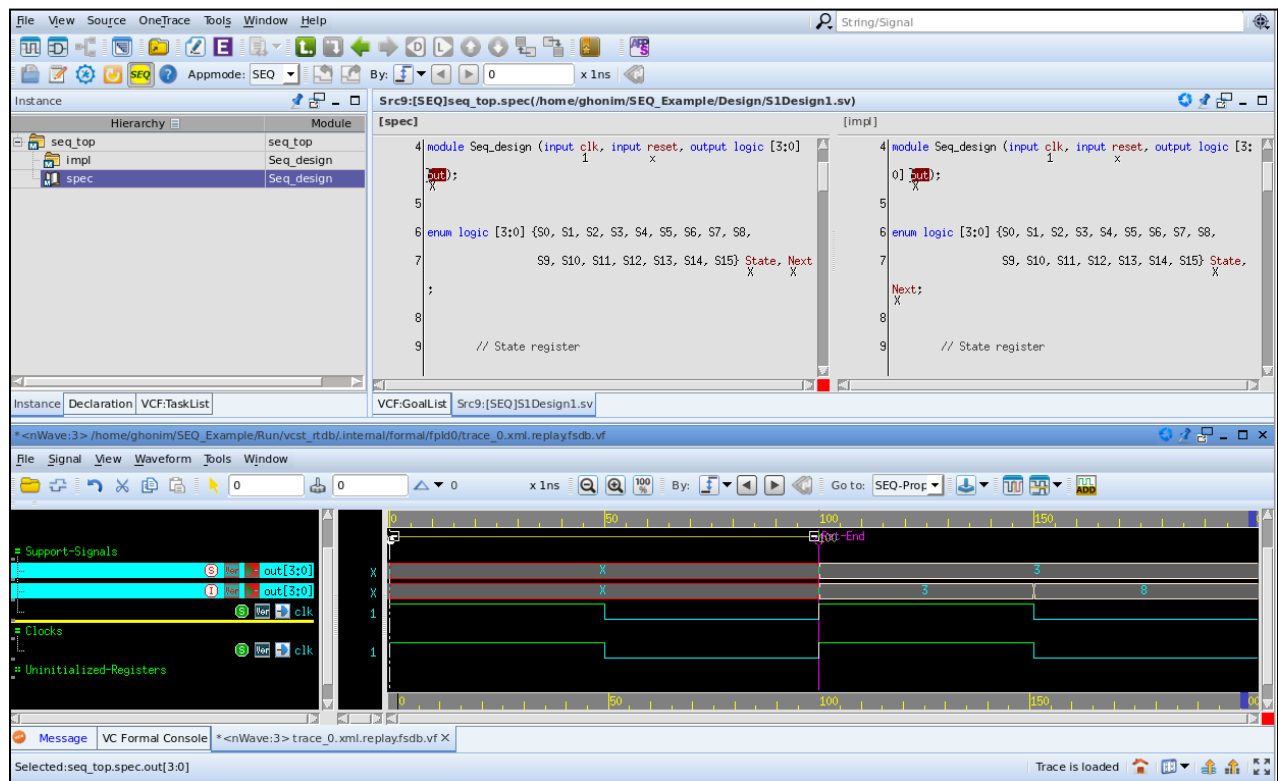


Figure 10. Examining the inequivalent parts of the design.

In Figure 10, in the black box on the lower left corner, we see “Support Signals” in green text, showing the output signals of the specification design (**S**) and the implementation design (**I**). The red color indicates the mismatch between the designs. We can also see that the outputs are not equivalent. To see where this is coming from, we can trace it back to its source by going back to the driving signal to that output:

As shown in Figure 11 below, under “Support Signals”,

- Right-click the implementation design (**I**) → Show OnceTrace Signals → Driver

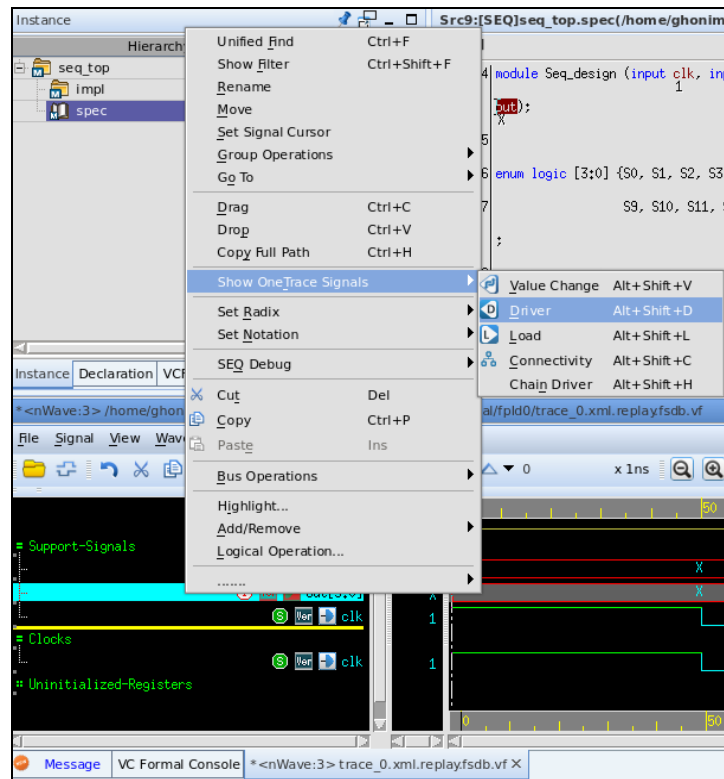


Figure 11. Tracing the source to find the discrepancy.

After selecting “Driver”, the Wave window should look like this:

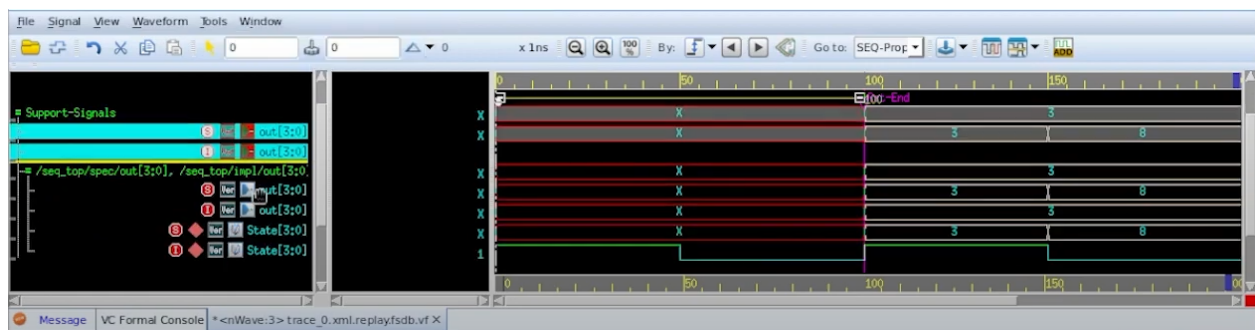


Figure 12. After tracing once, this is the waveform we get.

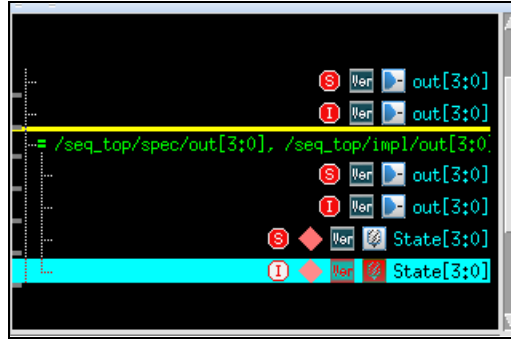


Figure 13. The states can also be seen as inequivalent.

We still need to trace this back more. Keep repeating the procedure above multiple times, or use the shortcut **Alt+Shift+D**, until we see a more specific source of the inequivalence.

Here are the manual instructions again:

- Right-click the implementation design (**I**) → Show OnceTrace Signals → Driver

Eventually, we will see that the mismatch is in the reset signal. There is also an “x” sign under it on the elaborated spec and impl codes as shown below.

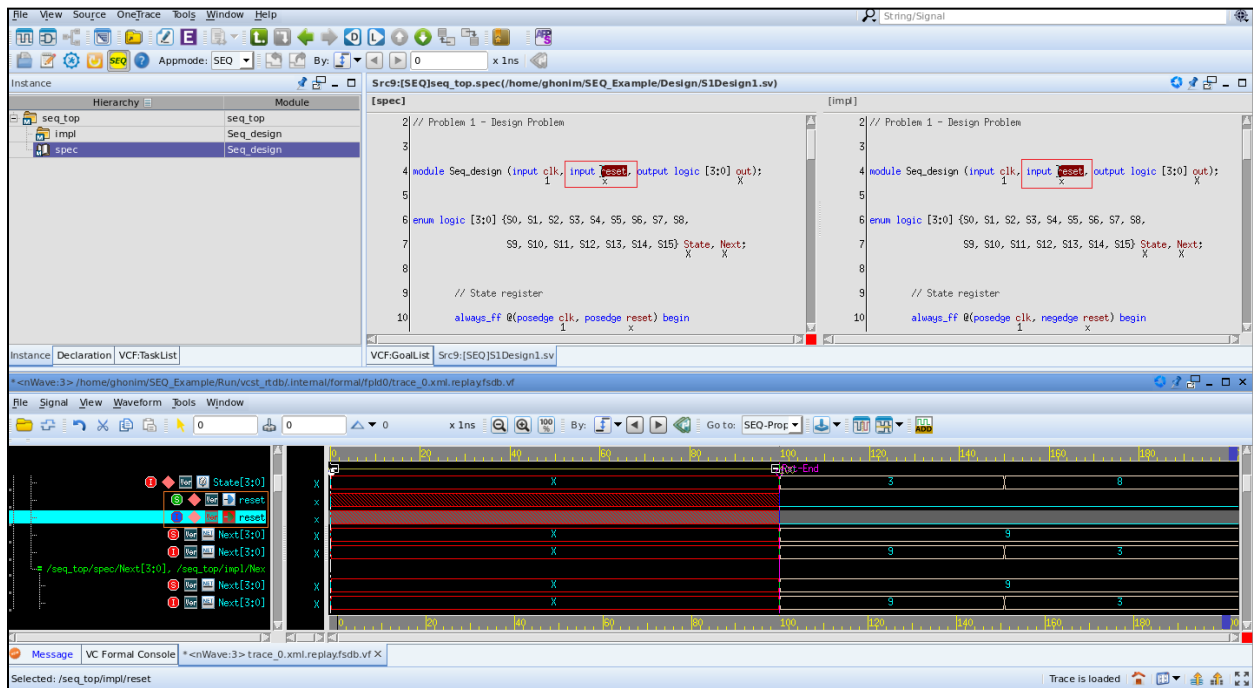


Figure 14. After multiple repeats of tracing. We can also see exactly where the errors are in the design files in VC Formal.

Resolving Errors

To fix this issue, we check the two design files and look at the [reset signal](#) in the Finite State Machine (FSM) implementation. Remember in the beginning of this tutorial, we noted that the implementation design has a negative edge-triggered reset, while the specification design has a positive edge-triggered reset.

Alter the implementation design and switch the **negedge reset** to a **posedge reset**. Now both the specification and implementation files should have the same reset.

Save the file, and go back to VC Formal.

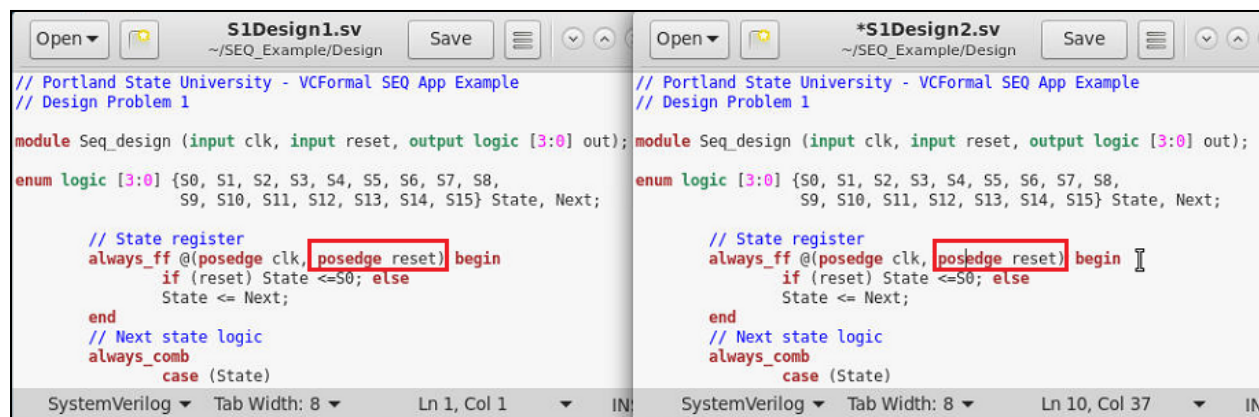


Figure 15. Altering the S1Design2.sv implementation file to have a posedge reset.

Before we load the TCL script again, we need to restart VC Formal by clicking on .

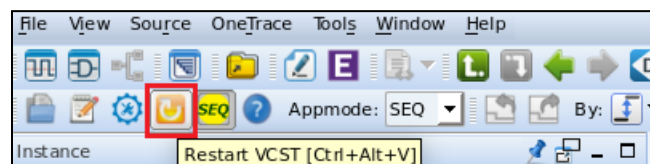

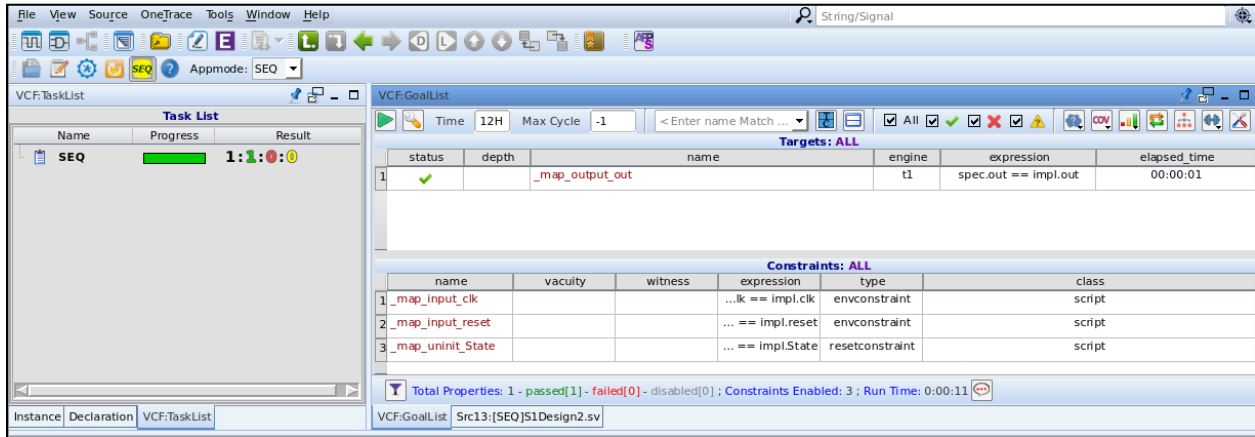



Figure 16. Restarting VC Formal.

Now load the TCL script and run the simulation as we did before. This time, we can see that the specification and implementation designs are equivalent through the  icon.




Task List

Name	Progress	Result
SEQ		1:1:0:0

VCF:GoalList

Time: 12H Max Cycle: -1

Targets: ALL

status	depth	name	engine	expression	elapsed_time
1		_map_output_out	t1	spec.out == impl.out	00:00:01

Constraints: ALL

name	vacuity	witness	expression	type	class
1 _map_input_clk			...lk == impl.clk	envconstraint	script
2 _map_input_reset			... == impl.reset	envconstraint	script
3 _map_uninit_State			... == impl.State	resetconstraint	script

Total Properties: 1 - passed[1] - failed[0] - disabled[0] ; Constraints Enabled: 3 ; Run Time: 0:00:11

VCF:GoalList Src13:[SEQ]S1Design2.sv

Figure 17. Equivalence checking is successful.