Sequence Equivalence Check in Jasper tool - automatic setup

Filip Pawelec
filip.andrzej.pawelec@intel.com

Abstract

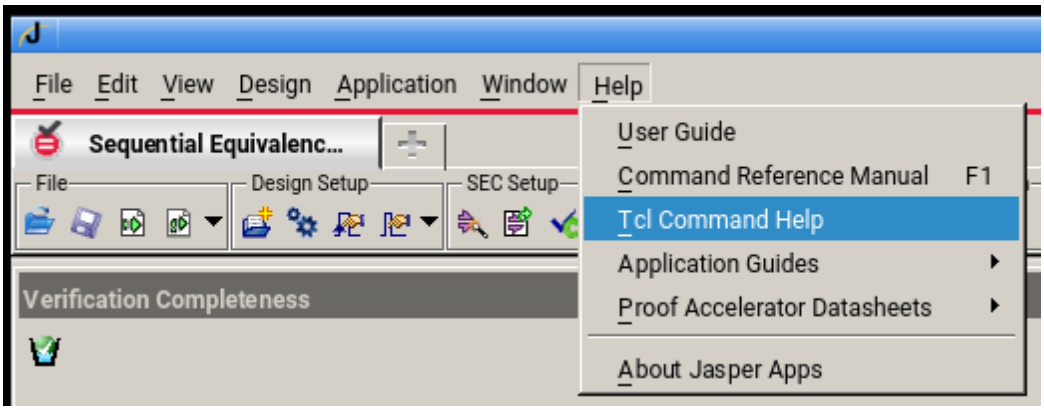
This document contains instructions necessary to work with dedicated tool for Sequence Equivalence Checking from Jasper apps suite. After finishing this exercise, student should be able to understand how to perform basic setup of the tool, as well as debug potential issues that may arise.

Kraków, Poland
June 14, 2024

Contents

DISCLAIMER

If during this laboratory at any point You're having trouble with TCL commands specific to Jasper, the best way to get help is to use the "TCL Command Help" tool available in Jasper:



If that fails, feel free to ask me questions, and maybe I'll be able to answer them :-)

1 Quick introduction to Sequence Equivalence Checking

At this point of the course You should be more or less familiar with the concept of Sequence Equivalence Checking, so if You feel that You got the grasp of the basics, feel free to skip this section entirely.

So, what is Sequence Equivalence Checking? In simplest terms, SEC allows us to answer the question: is module A output equivalent to module B output, given the same stimuli? In practice, often in order to answer this question we must make additional assumptions and/or simplifications on both the SPEC (specification – module that we treat as our golden model) and IMP (implementation – the module that we want to make sure did not bring any unexpected changes) design sides.

When SEC can be useful?

Example verification use cases may be one of the following:

- Clock gating insertion,
- Re-partitioning and pipeline re-timing,
- Code cleanup,

- Features addition / removal,
- ECO insertion,
- Parametrization,
- CPU lockstep verification,
- Reset optimisation,
- Prototyping.

2 Understanding the tool setup

The main difference in automatic vs. manual SEC is the setup needed. Manual setup should give the same results as the automatic one, if performed correctly. Main advantage of automatic setup is the time needed to compose a functional flow. The reason that it is good to know how to do the SEC manually is that not always You will have access to appropriate dedicated tool for automating this task. Having that in mind, let's dive into the setup of automatic SEC flow in Jasper.

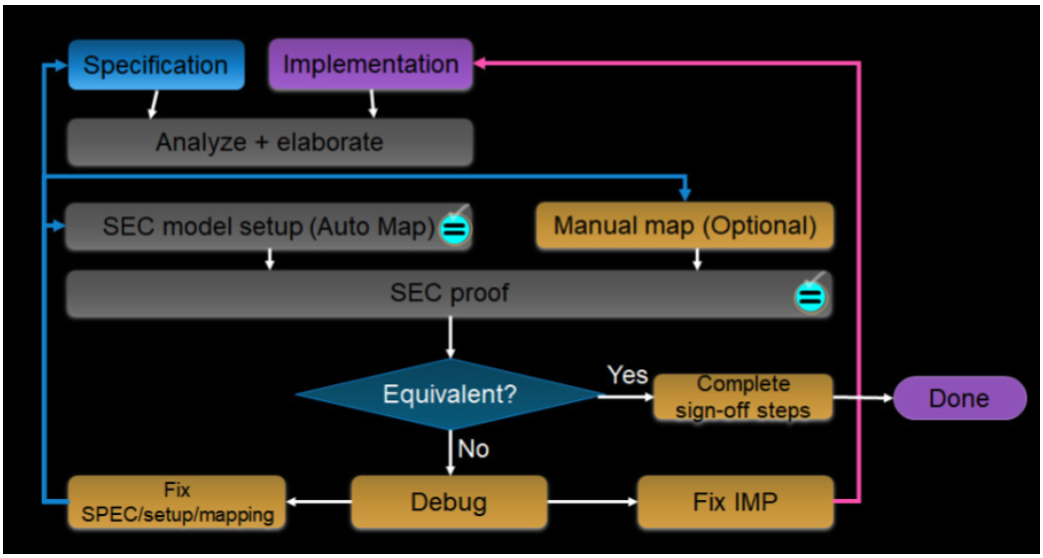


Figure 1: Typical SEC workflow in Jasper [Jasper Sequential Equivalence Checking App User Guide]

2.1 Analyze & elaborate

There is one command that is quite important to put at the beginning of each SEC script in Jasper - and that is the ***check_sec -clear all*** command. Also don't forget to include the ***clear -all*** command - those two commands will ensure that You start with a clean environment every time You rerun the flow, avoiding errors / warnings that may arise from inherited settings. After that, the first step in the flow setup is analysis and elaboration of specification and implementation sides of the design (the order is important!). There are actually three ways You can do that in Jasper:

Explicit way : Uses commands: ***check_sec -analyze -spec ...***, ***check_sec -elaborate -spec ...***, ***check_sec -analyze -imp ...***, ***check_sec -analyze -imp ...***

Context-driven : Uses commands to switch between context, and then uses typical analyze / elaborate commands: ***check_sec -compile_context spec***, ***check_sec -compile_context imp***, ***check_sec -compile_context fpv***

Joined setup : Uses command: ***check_sec -setup -spec_top <mod_name> -spec_analyze_opts {<opts>} -spec_elaborate_opts {<opts>} -imp_top ...***

Choosing the way You do it mostly comes down to preference, but for the purpose of this laboratory, we'll be using the joined setup method, as that keeps everything in one place. In the TCL script used for this laboratory, it will look similar to this:

```
check_sec -setup \
          -spec_top spec_dut_toplevel \
          -spec_analyze_opts { -sv12 -f spec_files.f } \
          -spec_elaborate_opts { -bbox_mul 128 -bbox_div 128
   -bbox_a 65536 } \
          -imp_top dut_toplevel \
          -imp_analyze_opts { -sv12 -f impl_files.f } \
          -imp_elaborate_opts { -bbox_mul 128 -bbox_div 128
   -bbox_a 65536 }
```

2.2 Signal mapping, waiving and tie off

In the ideal situation, the delta between the designs that we want to put through is small enough, that all the mapping is done completely automatically. Unfortunately, that is not always the case - sometimes You, the engineer, must specify the correlation (or lack there of) between signals in both designs.

There may be many situations where the tool may report something amiss with the mapping, to name a few:

- Signal names change (but the same functionality),
- Non-mutually existing ports on the designs,
- Bus bit width mismatch,
- Unmatched internal non-resettable registers,
- Floating internal wires,
- Black-boxed instances, wires, etc.

Jasper lets You handle such situations in different ways - You can waive such mapping issues, You can map them manually, You can specify the value of the undriven signals, specify value of non-resettable registers, and many more. Correct mapping is very important, because it directly affects the validity of the proof that You're going to get. Generally speaking, the command to perform operations on unmapped signals is the ***check_sec -map*** command. It has a ton of options,

so there is no point of going through all of them here (and there is no "one size fits all" solution on how to use them). Here are some example usages of this command:

```
# To map signals manually
check_sec -map \
   -spec {af_vld af_data} \
   -imp {affine_valid_c affine_data}
# To map signals semi-manually
check_sec -map \
   -spec {{[0-9]_vld_\w+} {aff_[0-3]_(data|data_last)}} -regex \
   -imp {{id_no_[0-9]_valid} {affine_[0-3]_(indata|indata_last)
   }} -regex
# To indicate that some signal from imp is delayed by a const
   num of cycles
check_sec -map \
   -spec {some_signal} \
   -imp {some_signal_2} \
   -imp_delay 3
```

For exact usage of this command in a given use case, it is best to consult SEC app guide and "TCL Command Help" tool.

When there are some signals that cannot be mapped to each other in any way (we, for example, inserted a new feature and deleted one old, both tied to some additional signals), we can waive those unmapped signals:

```
check_sec -waive -waive_signals \
   {dud2in simple_cnt_module_imp.dud_out}
```

2.3 Clock & reset setup

In this step, we must provide the tool with information about clock and reset structure. In complex designs with different clock domains and clock gating that task alone can be a bit daunting, but that is not the focus of this laboratory. Our exemplary design fortunately only has one clock and one reset signal:

```
reset -expression !(nreset)
clock clk
```

2.4 Prove and sign-off commands

After we have done all the prep work for our SEC flow, all that remains is adding prove and sign-off commands. One very handy command to use in this flow is ***check_sec -interface***. This command performs a sanity check on signal mapping, ensuring that everything is done accordingly. If there are some mapping inconsistencies, it is best to focus on them first before continuing with the debug. To run

the actual proof, we'll use the ***check_sec -prove*** command. This command will give us all the results in the graphical form in GUI. However, it is always a good practice to have results in a form of a text report file for any future reference - we can generate such report using ***check_sec -signoff*** command with appropriate switches.

3 Running the tool flow & GUI basics

Now that we have talked about the most important aspects of the SEC flow, we can run our script and look at the results:

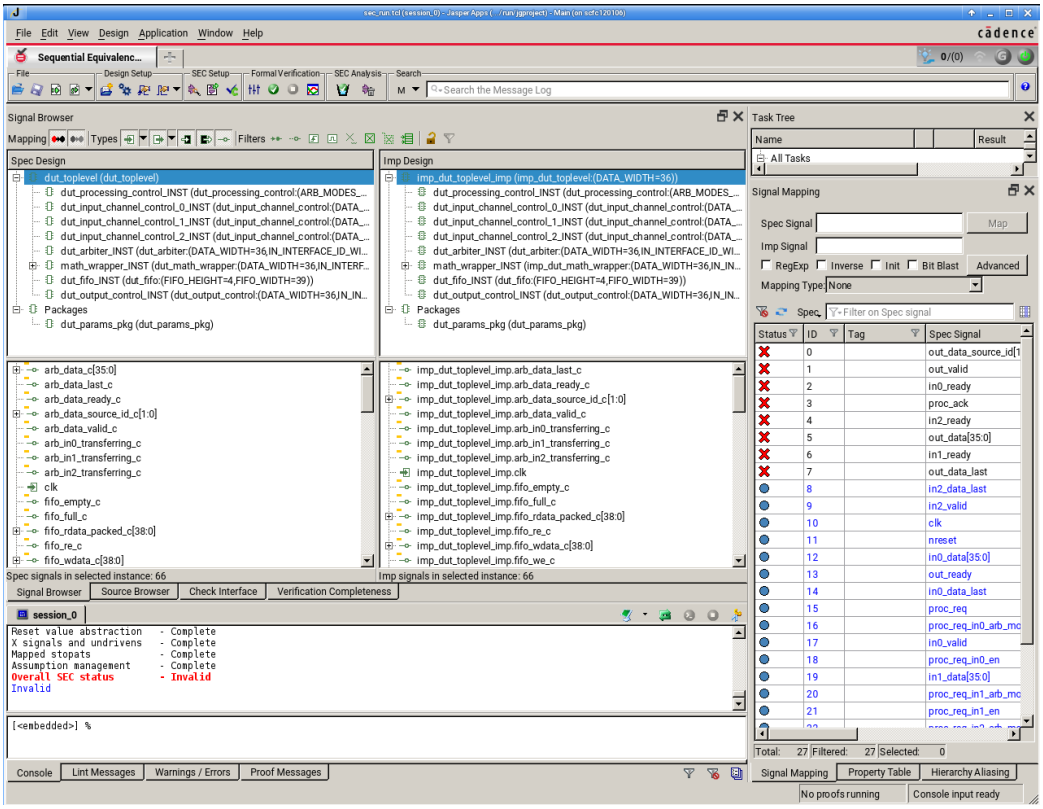


Figure 2: Default GUI view for SEC in Jasper

If we run Jasper from a UNIX shell with a switch ***-sec***, the above is the default look for SEC app GUI. The middle section, with 4 adjacent windows, is the signal browser, that allows You to compare signals in both spec and imp, on all hierarchy levels. Please note the 4 tabs that are just below the signal browser - **Signal Browser, Source Browser, Check Interface, Verification Completeness**. The two most useful tabs are the "Check Interface" tab and "Verification Complete-

ness" tab. In "Verification Completeness" tab, we have all the relevant data from our tool run collected - mapping, boundary proof, etc. In "Check Interface" tab You'll find all the mapping performed, its status and validity. On the right side of the GUI window, we have all the properties tied to boundary signals in spec and imp. You are encouraged to play around a bit with the GUI - just don't change any settings that are permanent.

4 Debug & tasks

You are given two nearly identical designs, simulating two points in time in project repository, where the changes between those two snaps are not too big. Your main objective is to use the debug capabilities of Jasper SEC app to find the mistakes and fix them. In case of any questions, I am here for You - please ask away. Good luck!