ASIC DESIGN

Debugging Formal Verification (FV) Problems .. FV Primer

MAY 13, 2012 | KBULUSU | 1 COMMENT |

Anyone who has been around for a while in the FV (formal verification) world for sometime agrees that FV is easy as long as the FV environment is setup correctly. Ofcourse when FV says failure , it is a whole different issue . BTW, I'm talking about formal verification for Digital Ckts. Will cover FV for analog circuits later.

Just for the record, Most of us in the industry or most companies which develops FV tools, when they say formal verification , they mean equivalence checking. There is another type called model checking as well .Search google if you want to know more about model checking as well.

This is definition of Formal verification according to wikipedia :

http://en.wikipedia.org/wiki/Formal_verification (http://en.wikipedia.org/wiki/Formal_verification)

There are many people who dont understand when FV has to be used and when functional simulations have to be used. I met one person who thought FV replaces the simulations . So, I just want to clear things first. Once the RTL coding is done, you might want to check if the written RTL conforms to the spec and is functionally correct without any glitches etc. Once this RTL simulations (RTL functional simulations) pass , the design is then synthesized .

You want to do formal verification to answer the following question : Is the synthesized circuit functionally same as intended in the RTL? Yes, you can still do gate level simulations and verify the same. But simulations are slow and you need to do exhaustive simulations to cover all the possible vectors for a given circuit. Formal Verification can do the same in less time mathematically .

I'm are not going to talk about the different types of simulations and pros/cons of each here. Similarly I'll not talk about assertion based simulation either. There are tons of articles and lecture notes on the web .

Formal verification is still a niche area and less than 25% of the design companies are using formal verification today. Partly because the Formal tools are either costly, takes time to setup the env and there arent enough resources for FV.

From the methodology perspective, there are two methodologies hierarchical and flat based.

Hierarchical Methodology : It is very fast and verifies the design by partitioning it based on the hierarchy. It is very similar to the bottom-up approach in synthesis. Once the lower modules/blocks are verified, it blackboxes them and then verifies the next upper level in the hierarchy. Once all the modules/blocks are verified and blackboxed, it then verifies at the top level (glue logic). There are some inherent issue that comes with this methodology : there are can be some false failures because of the blackbox approach. If there are cases when the constants have to be propagated from the top level, then the tool will not be able to do so and hence it might be flagged as false failures. It is not problem with the FV tools, but inherently a draw back with the methodology itself.

Flat Level Methodology : If the design is flattened (hierarchy is collapsed) in synthesis,this is the only way one can verify the circuit.The runtimes can vary depending on the design size and complexity and the tool ofcourse.

Having talked about the FV methodologies, lets talk about two important pieces of Formal Verification:

Setting up Formal Verification Environment : This is not a straight forward process . Some of the things that have to be noted are :

Talk to your logic/physical synthesis engineer . Yes, without his inputs, you will never know what options/configurations he has used in synthesis and without this knowledge, you will never be able to let your FV tool what the synthesis tool has done in terms of optimizations. You need to mimic the synthesis environment as much as possible.

At the minimum , you need to know the following :

1. Any Pragmas that have been used in RTL like compiler directives , translate_off/on, full_case/parallel_case , async_set_reset etc

2. If specific value is used for Dont-care optimization .

3. If constant propagation is enabled and used

4. If any logic is marked as set_dont_touch or force kept

5. If any ports/nets/pins are tied to constants.

6. If you are comparing pre and post DFT netlists, then you need to disable the scan logic by driving scan_enable/test_mode ports to a constant value . so, get this information.

7. If any models doesnt have the definition available or if any model is assumed or modeled as blackbox in synthesis, you need to get that list and blackbox the same .

8. If clock gating is enabled in synthesis , you need to get the clock gating configuration (name of clock gating cell used in synthesis) ;

9. Get the list of libraries used in synthesis ; it is recommended to get and read the .lib into the FV tool (assuming FV tool can read .lib directly) rather than using the verilog netlist written out of synthesis tool. Some synthesis tools dont write the complete definition of a model and they just write the interface for the module. If you dont have .lib files, at the minimum , verilog simulation models need to be read in.

10. Designware : Get the list of the designware components used in the design. There are some limitations as to what designwares can be formally verified . For example, designware multipliers , ECC etc cant be formally verified and has to be simulated to check the functionality.

11. Re-timing : Check if re-timing is enabled in the synthesis ; some synthesis tools automatically detect and perform retiming when some datapath components are used.

12. Clock gate cloning : If clock gate cloning is done, some FV tools require you to explicitly mention the list of all clock gate cells that got cloned and their names. Otherwise you might see false failures.

13: Sometimes synthesis tools synthesize sequential elements whose state is inverted with respect to the RTL ; so it is the case, you need to enable that option in the FV tool.

14. Naming Convention: Most synthesis tools follow a standard naming conventions and sometimes your FV tool might not be able to map the element/object in the reference netlist to implementation netlist. so, get the naming conventions in your synthesis tool and configure your FV tool accordingly.

15. check if there is any default value set for undriven nets/pins/ports.

16. Some advanced options like if d-input of the flop is/has tied to be constant (1 or 0) or if the clock has been/need to be tied to constant high or low, enable the relevant configs.

17. check if there are datapath components in the design. Sometimes, if the size of the RTL inferred multipliers is more than 40bit, then some special datapath solving techniques might be necessary. Also there is a limitation on this FV technology on how much wide multipliers you can verify. If you believe you might hit the limit,then better blackbox that component. A 128bit multipler can takes ages to get it verified.

18.Port mapping/aliases : Sometimes during CTS , the physical synthesis tools while building clock-tree/reset-tree networks , they might duplicate the ports (for example there might be a reset port called “rst” and tool might have created rst_l port as well which is logically and functionally equivalent to “rst”) . So, if you are doing FV between pre-CTS and post-CTS netlists, you need to take care of this either by port mapping or through some aliasing features in the tool.Else the tools might report false failures.

Debugging Formal Verification Issues: These are some tips and by no means a complete guide. Debugging comes by experience and the more issues you solve, the more insight you will have. Some of the bullets below might look like a repeat of the points I mentioned in the above section “Setting up formal verification environment” .

General suggestion : Formal Verification can be done between RTL and synthesis netlist or between synthesis netlist and DFT netlist etc, depending on the stage and flow you are in. It is always recommended to verify the circuit in increments between two immediate stages in the flow. If you hit a FV failure , it will be very difficult to narrow down which stage in the flow has caused this and which optimization step or command is causing this.

Also when a formal tool raises a flag, it is recommended to first do an simulation between reference(Golden) and implementation netlist first. Most formal tools generate test vectors for the failing compare point and you can use the same to simulate and see if the simulation results match for reference and implementation. If they match, then it means it is a false failure and you know that your synthesis tool is correct.

Some formal tools have very advanced capabilities like generating a test bench for a given logic cone. So, if you are ever caught between two formal tools and they disagree on any result, then generate the test vectors and run functional simulation using it. Some formal tools dont generate test vectors (they use vector less approach), then get the logic cone on which the formal tool is complaining and generate the testbench for that logic cone in the other formal tool and then run simulations.

casez and casex statements : Somes times synthesis tools infer the casex/casez statements and based on the dont care optimization alogorithsm, they might create a different logic than the formal tool . For example, if there is an if – else loop and if the variable is not declared earlier , but used in the loop, synthesis tool might not always infer a latch if it feels that there is no necessity to store a state. What I would suggest is, check for 2 cycles and see if there is a necessity, if not, then synthesis tool is correct as latch is redundant here.

Pragmas: Make sure the intepration of pragma’s between synthesis , formal and simulation tool is same. Some simulators like ncsim doesnt support certain synthesis pragma’s like translate_off/on and so they might interpret the logic and gives incorrect results when you want to see if your formal tool is flagging an false error.

Also, in most of the cases , formal errors get introduced during the rtl elaboration stage and certain RTL constructs like generate statements, genvar, multi-dimensional arrays etc can confuse our little buddy (rtl elaborator) . So check for these advanced or complex constructs in RTL.

Then check for if any variables are used but never given a value, etc . These are some corner cases though 🙂

Lastly, formal errors come from very small snippet of the RTL code. So, using divide and conquer approach, reduce the design to as small as possible while maintaining the same formal erorrs on the same compare points/logic cone.

If, you see thousands of failing compare points, you’d better check your SETUP to see if there is anything wrong. Unless the synthesis tool is developed by people who dont understand it, it is not very common to see thousands of failing points for a decent sized testcase (300k cell count) . If you do see,then my adivce would be , STOP USING THAT SYNTHESIS tool 🙂

One thought on “Debugging Formal Verification (FV) Problems .. FV Primer”

1. Pingback: Debugging Formal Verification Problems : Part II | Kiran Bulusu Blog on Semiconductor/EDA domain Edit