Gate level simulations: verification flow and challenges

Ankit Khandelwal, Abhinav Gaur, Deepak Mahajan, - March 05, 2014

Introduction

Simulations are an important part of the verification cycle in the process of hardware designing. It can be performed at varying degrees of physical abstraction:

(a) Transistor level
(b) Gate level
(c) Register transfer level (RTL)

In many companies RTL simulations is the basic requirement to signoff design cycle, but lately there is an increasing trend in the industry [1] to run gate level simulations (GLS) before going into the last stage of chip manufacturing. Improvements in static verification tools like Static timing analysis (STA) and Equivalence Checking (EC) have leveraged GLS to some extent but so far none of the tools have been able to completely remove it. GLS still remains a significant step of the verification cycle footprint. This paper focuses mainly on the steps to run GLS from verification perspective and the challenges faced while running it.

Firstly, let us look at the reasons why GLS is still used in the industry despite all the challenges associated with its execution.

Motivation for running gate level simulations

The main reasons for running GLS are as follows:

1. To verify the power up and reset operation of the design and also to check that the design does not have any unintentional dependencies on initial conditions.
2. To give confidence in verification of low power structures, absent in RTL and added during synthesis.
3. It is a probable method to catch multicycle paths if tests exercising them are available.
4. Power estimation is done on netlist for the power numbers.
5. To verify DFT structures absent in RTL and added during or after synthesis. Scan chains are generally inserted after the gate level netlist has been created. Hence, gate level simulations are often used to determine whether scan chains are correct. GLS is also required to simulate ATPG patterns.
6. Tester patterns (patterns to screen parts for functional or structural defects on tester) simulations are done on gate level netlist.
7. To help reveal glitches on edge sensitive signals due to combination logic. Using both worst and best-case timing may be necessary.
8. It is a probable method to check the critical timing paths of asynchronous designs that are skipped by STA.
9. To avoid simulation artifacts that can mask bugs at RTL level (because of no delays at RTL level).
10. Could give insight to constructs that can cause simulation-synthesis mismatch and can cause issues at the netlist level.
11. To check special logic circuits and design topology that may include feedback and/or initial state considerations, or circuit tricks. If a designer is concerned about some logic then this is good candidate for gate simulation. Typically, it is a good idea to check reset circuits in gate simulation. Also, to check if we have any uninitialized logic in the design which is not intended and can cause issues on silicon.
12. To check if design works at the desired frequency with actual delays in place.
13. It is a probable method to find out the need for synchronizers if absent in design. It will cause “x” propagation on timing violation on that flop.

**Execution strategy for GLS**

1. Planning the test-suite wisely to be run in GLS

In highly integrated products it is not possible to run gate simulation for all the SoC tests due to the simulation and debug time required. Therefore, the list of vectors which are to be run in GLS has be selected judiciously. The possible candidates for this list are:

- Testcases involving initialization, boot up.
- All the blocks of the design must have atleast one testcase for GLS.
- Testcases checking clock source switching.
- Cases checking clock frequency scaling.
- Asynchronous paths in the design.
- Testcases which check entry/exit from different modes of the design.
- Dedicated tests for timing exceptions in the STA.
- Patterns covering multi clock domain paths in the design
- Multi reset patterns are a good candidate for GLS

It must also be made sure that the test cases selected to be run in GLS are targeting the maximum desired operating frequency of the design. Should there be no time constraints, all tests run in RTL simulations can be rerun in GLS. Also, if there are no tests fulfilling the criteria mentioned above, then they should be coded.

2. Preserving design signals

Some signals which are critical for GLS debug can be preserved while synthesis.

3. Testbench updates for GLS

A list of all the synchronizer flops is generated using CDC tools. Also, other known asynchronous paths where timing checks needs to be turned off are analyzed and complete list of such flops is prepared which also includes reset synchronizers. The timing checks are turned off on all such flops to avoid any redundant debugging, otherwise they will cause “x” corruption in GLS. This work should be ideally done before the SDF arrives. It may happen that the name of the synchronizers in RTL and the netlist are different. All such flops should be updated as per netlist. Also, the correct
standard cell libraries, correct models of analog blocks, etc. need to be picked for GLS.

4. Initialization of non resettable flops in the design

One of the main challenges in GLS is the “x” propagation debug. X corruption may be caused by a number of reasons such as timing violations, uninitialized memory and non resettable flops. There are uninitialized flops in design which by architecture are guaranteed to not cause any problems (if they settle to any value at start). Thus, there is a need to find out all such flops in the design (which are reviewed with designers) and initialize them to some random value (either 0 or 1) so as to mimic silicon.

5. Unit delay GLS for testbench cleanup

This step is not compulsory but is generally very fruitful if employed in the GLS execution. The setup is done for unit delay GLS (no SDF) and the testcases that are planned to be run on gate level are run with this setup to clean the testbench. This is done because the unit delay simulations are relatively faster (than those with SDF) and all the testbench/testcase related issues can be resolved here (for example - change probed logic hierarchy from rtl to gate, wrong flow of testcase, use of uninitialized variables in the testcases that can cause corruption when read via core, etc). Running the unit delay GLS is recommended because one can catch most of the testbench/testcase issues before the arrival of SDF. After the SDF arrives, focus should be more on finding the real design/timing issues, so we need to make sure that the time does not get wasted in debugging the testcase related issues at that time. Annotation warnings cleanup on SDF

6. Annotation warnings cleanup on SDF

When the SDF is delivered to verification team, then the simulations are run with the SDF/netlist. Specific tool switches need to be passed which picks us the SDF and tries to annotate the delays mentioned in the SDF to the corresponding instances/arcs in the netlist. This is known as “back-annotation.”

During this process, many annotation warnings are encountered which need to be analyzed and either sorted out or waived off by the design team. Most important warnings which need to be looked into are the ones which are due to non-existent paths i.e. the SDF has an arc but netlist does not have it. Also, there can be mismatch between the simulation model “specify” block and .lib of the IP.

7. Running GLS with SDF:

(a) Early SDF (for initial debug)
This step involves use of a sdf in which timing is met at a lower frequency than target and GLS can be run at that frequency. This helps in cleaning up the flow and finding certain issues before the final SDF arrives.

(b) Full speed SDF (WCS and BCS)
This step starts with the sdf in which timing is met at target frequency. The entire setup is done and the planned pattern-suite is run on this setup and failures needs to be debugged according to a priority list which should be made before hand. All the high priority patterns need to be debugged first.

All issues found are discussed with the design and timing team and the required fixes are done in the netlist/SDF. This process is repeated until the GLS regression is clean on the final SDF.
The following diagram shows the GLS sign off flow:

**Challenges faced during GLS setup**

GLS with SDF delays annotated is usually very slow and time consuming. A lot of planning is needed to wisely select the patterns that need to be simulated on netlist. Run-time for these tests and memory requirements for dump and logs must also be considered. Identifying the optimal list of tests to efficiently utilize GLS in discovering functional or timing bugs is tough. These are to be decided with discussions with timing team and design team on which paths are timing critical/asynchronous and needs to be checked.

GLS is much more difficult than RTL simulations as delays of gates, interconnects comes into picture and RTL assumptions for arrival of data/clock may mismatch causing failures.

During the initial stages of GLS, identifying the list of flops/latches that needs to be initialized (forced reset) is a big hurdle. The pessimism in simulations would cause “x” on the output of all such components without proper reset, bringing GLS to a standstill. If approved by design and architecture team, these initializations should be done to move forward.

During synthesis, the port/nets may get optimized/re-named to meet the timing on these paths. Monitors/Assertions hooked in testbench during RTL simulations need to be revised for GLS to make sure the intended net is getting probed. Also, in netlist there can be inverters that can be inserted on that signal driver due to boundary optimization and can cause false failures (not having any design issues). This is really tough to figure out and debug.
If the gate level simulation with SDF is done without a complete synchronizer list, then failure debug to find such cases on gate level is quite cumbersome. Multiple debug iterations may happen in GLS to find out many such flops.

Debugging the netlist simulations is a big challenge. In GLS, models of the cells make the output “x” if there is a timing violation on that cell. Identifying the right source of the problem requires probing the waveforms at length which means huge dump files or rerunning simulations multiple times to get the right timing window for violations. The latest tools are offering “x” tracing techniques for quickly tracing the source of “x” propagation. Such tool features need to be explored. All said and done, it can be concluded that one requires a lot of patience while debugging the GLS waveforms.

**Conclusion**

In short, despite being a time consuming activity and having many challenges in setup and debug, GLS is a great confidence booster in the quality of the design. It can uncover certain hidden issues which get missed out or difficult to find by RTL simulations. This is the reason many organizations prefer this activity before tape-out because it is more close to the actual silicon and gives a fairly good idea about how the design will behave in real. The probability of having sound sleep after tape out improves with GLS.

**References**

3. Logic simulation, Wikipedia.
4. Jalan, Gaurav, "[Gate Level Simulations : A Necessary Evil - Part 2](http://whatisverification.blogspot.in)," whatisverification.blogspot.in, June 29, 2011.