

Method to Partition Your Gate Simulation Debug

Kendall Chan

AMD
Markham, Ontario, Canada

www.amd.com

ABSTRACT

Full-chip system on chip (SoC) gate simulations are a challenge for design verification teams. Long compile and run times, large memory host requirements, and difficulty with signal visibility make verification debug challenging, though this effort does pay off when you catch the occasional design bug.

A technology from Synopsys called Siloti What-If-Sim-Replay allows you to partition the gate simulation task across the design. The partitions are small in memory footprint and can simulate fast. This paper presents a user experience of how the Siloti What-If-Sim-Replay technology was used in a gate regression flow. The paper describes in detail the simulation harness created to compile and run gate simulations and to check the results.

Table of Contents

1.	Introduction.....	3
2.	Past strategies for improving gate simulation performance.....	3
3.	Gatesim debug challenges.....	3
4.	Tile gate simulation flow	5
5.	Background to wisim data-replay flow.....	8
6.	Issues to achieve ‘passing’ gate tiles through gate tile flow.....	11
7.	Results.....	12
8.	Conclusions.....	13

Table of Figures

Figure 1 : Tiling approach of ASIC reflected in top-level physical layer that divides design into tile partitions in RTL design through to physical design.....	4
Figure 2 – Example of synthesized logic that will simulate as X at flop input, but should resolve to a known value given that we have OR of a X and Xbar from a common source.....	5
Figure 4 : Scaling of design size and depiction of tile-level testbenches	6
Figure 5 : High-level view of per tile replay flow	7
Figure 6 : Feedthrough path with repeaters flops connecting an output of A to an input of B	7
Figure 7 : Depiction of processes, files, and libraries to run ‘wisim’ flow.....	9
Figure 8 – Incorrect behaviour through a flop stage due to simulation race when both D and CLK change at same time	12

Table of Tables

Table 1: List of resolved issues to achieve flow success	11
--	----

1. Introduction

Gate simulations for most ASIC verification teams are difficult due to slow compiles, long simulation run times, and limited waveform visibility for debug. The problem increases with netlist size as the netlist matures through the physical design process from a ‘pre-layout’ netlist, to a ‘post-layout’ netlist, to a ‘post-layout power aware gate netlist’. For an AMD graphics northbridge design (Gnb), VCS compile times for the post layout power aware gate netlist are over 24 hours and simulation runtimes for basic tests span a few days. Debug visibility in each gate simulation is limited due to the effects on simulation performance and process memory size. This all combines to make the debug of gate-level simulation failures very slow and iterative.

This paper describes a new and simple approach we’ve explored at AMD to partition our gate verification task into small ‘tile-level’ simulations that are fast to compile and run. We take advantage of the fact that the AMD Gnb designs utilize a tiling approach from RTL implementation through the physical design netlist implementation. We utilize a new waveform playback technology from Synopsys called Siloti What-If-Sim-Replay to provide the harness for us to replay RTL FSDB input stimulus onto each gate design tile. We then use the nCompare waveform compare utility to help us confirm our simulations are cycle accurate. This method does not provide 100% verification coverage, as will be described later, but it is a very good platform for us to prescreen the gate tile designs and their respective technology model components for most issues ahead of the full SoC/Gnb gate simulation effort.

2. Past strategies for improving gate simulation performance

Different approaches for improving gate simulation performance have been used in the past. The simplest option is to find fast, large memory machines. Teams have also mixed gate tiles into an RTL simulation environment to speed up verification on specific gate tile netlists. Some teams have also utilized Siloti waveform observability tools to improve waveform dumping efficiency. These techniques have helped, but it is still a large and increasingly difficult problem as the designs have scaled faster than the benefits these methods provide. Due to the current challenges, many teams have also taken a serious look at not running full-chip SoC gate simulations.

3. Gatesim debug challenges

Gate simulation teams typically work through a few challenges when bringing up their test environment and their design. A few such issues will be described.

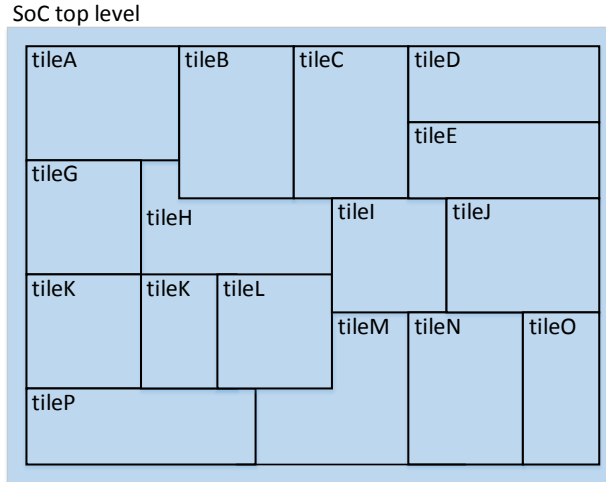


Figure 1 : Tiling approach of ASIC reflected in top-level physical layer that divides design into tile partitions in RTL design through to physical design

Long iterative debug process:

At the SoC level, gate simulation compile and runtimes can be extremely slow, and the required memory is large. As an example, compile times for AMD's graphics northbridge power-aware gate netlist will compile longer than 24 hours, and will simulate for typically 48 hours or longer. Both steps require over 100 Gb of memory. The gate simulation test debug effort is to find the specific issue in a particular design tile or in the test environment that causes the test to fail. There are likely many other bugs hiding behind this first issue, but they won't be visible until at least the current issue is resolved and the test is rerun. You iterate in this process to find the next bug until the test passes. It is a slow onion-peeling debug process, and by nature does not lend itself well to debugging issues across the design in a parallel fashion.

Testbench issues:

Many of the issues teams need to resolve with gate simulations are testbench related. There may be specific programming options or testbench forces that are required for the design to boot that may need to be adjusted with the transition from RTL to the gate view of the design. Typically, there are also many verification checkers that might reference internal signals in the design, which may need to be either fixed or removed to account for changes in the design signals that are visible in the gate netlist. Sometimes, there is also testbench logic that supports prefilling specific cache memories in the design which may need to be updated.

Netlist and technology library quality:

At AMD, verification of gate tiles often happens in parallel to the physical design process for the design tiles. Given the schedule pressure, the verification team will often try to simulate netlist views of design tiles that don't yet have logical equivalence confirmation or clean power rule checks signed off by the physical team.

Apart from concerns about the design netlist, there are also concerns about aligning the right technology libraries for the stdcells, analog macros, and memory macros. Some of these models and their respective views may only be first used in the gate simulation flow so they have not been tested in other SoC simulation environments.

Need to apply \$deposits to specific registers:

Another difficult area for the gate simulation effort over the last 10 years has been the need to apply many \$deposits into the gate design at specific register outputs. Synthesis tools are optimizing logic in such a way that makes them prone to propagate 'Xs' in simulation. With careful review, the user can often determine that the design source will logically resolve to a known value. In the simplest example, we could see cases where a register output is an unknown value in the gate simulation case, where as the same register in RTL may go to a known value after reset. In the gate case, the 'X' at the register output may actually feed back to be one of the drivers to the input of the same register; however, before it arrives the signal splits and gets inverted on one side and then 'OR'd'. In this case, you can infer that the 'OR' gate should logically output a '1', however a simulator would always simulate this as 'X'.

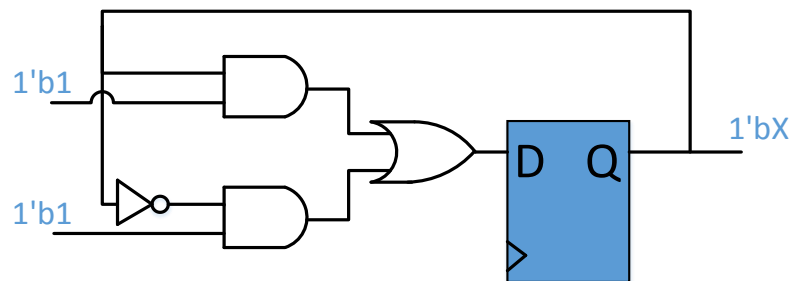


Figure 2 – Example of synthesized logic that will simulate as X at flop input, but should resolve to a known value given that we have OR of a X and Xbar from a common source

SoC-level gate simulation efforts will pend till all design tiles are ready:

The physical design flow for a large SoC has many tiles going in parallel across many team members. Different tiles will typically complete at different stages, but current SoC teams must wait for all tiles to be ready as a complete set before the SoC effort can really start.

4. Tile gate simulation flow

The approach described here is to address these gate simulation problems in a simple divide and conquer approach. A large ASIC design netlist typically uses a tiled approach during physical design floor planning. Each tile is a unique puzzle piece to the rest of the design. This tile structure reflects back into the RTL design environment. The input and

output ports of each tile in the RTL implementation of a tile match the same ports of the gate-level tile.

The tiled nature of the gate netlist designs is usually reflected in the fact that the source for each tile will sit in separate files. This makes it easy to compile along with its respective technology libraries. The challenge for a tile-level verification flow is to define a testbench that can provide valid stimulus for a variety of tests and validate the corresponding outputs. It turns out the solution is fairly simple by utilizing Synopsys Siloti What-If-Sim-Replay for FSDB waveform replay and by utilizing the nCompare waveform comparison utility.

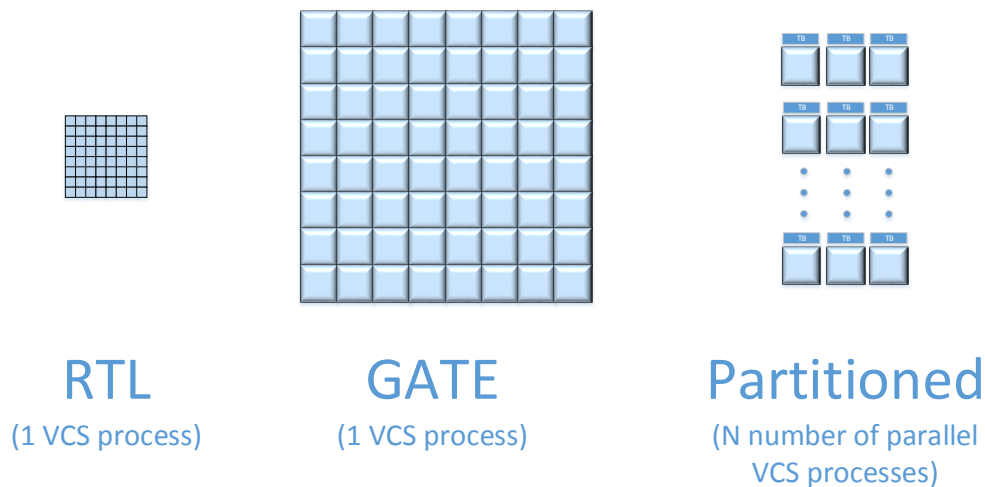


Figure 3 : Scaling of design size and depiction of tile-level testbenches

Through Synopsys Siloti What-If-Sim-Replay technology (we'll refer to this simply as 'wisim'), we can split the verification task for each individual tile into its own autogenerated FSDB replay testbench. We start with a passing RTL reference test FSDB to define the sequence of input values for each tile of the design from time 0 as well as to define the expected output values for each tile of the design. We setup the wisim to replay the RTL reference FSDB on a per tile basis.

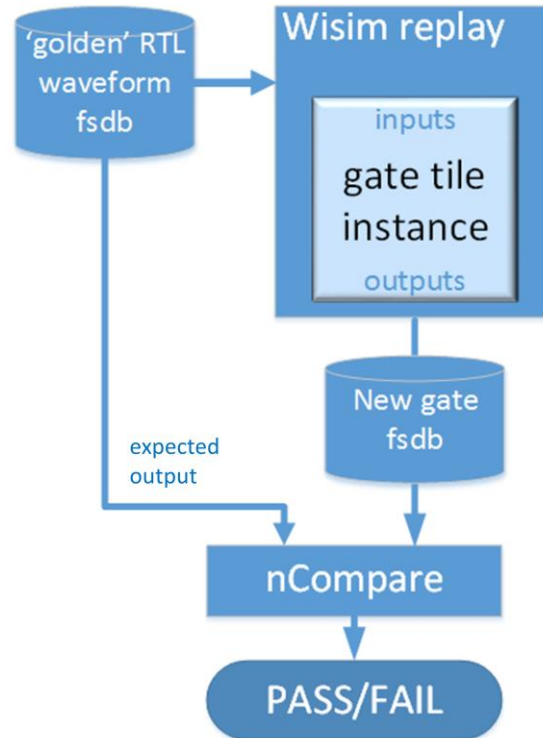


Figure 4 : High-level view of per tile replay flow

There are some assumptions we make in the setup for each tile. We connect top-level power ports, we buffer out repeater flops (which will be reviewed later), and we ignore the many feedthroughs (in the case of a post layout netlist) that traverse through the tile. Feedthrough logic passing from a tile input across to a tile output should have no logical effect on the other logic internal to the tile.

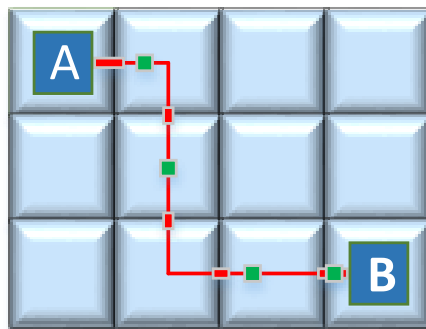


Figure 5 : Feedthrough path with repeaters flops connecting an output of A to an input of B

The wisim-gate tile replay simulation creates its own waveform FSDB output. This must be cycle accurate to our RTL reference FSDB assuming the RTL and gate designs are functionally equivalent. In the AMD testcases, there were a few changes required to achieve cycle accuracy to the RTL reference in the gate replay simulation which will be reviewed a little later in this paper. We use the nCompare waveform compare utility to

compare the replayed tile output waveforms against the same output signals in the RTL reference waveform. There can be some small timing skew as there are typically small timing delays in the gate stdcells, but the nCompare utility can accept a user defined time window tolerance to filter these out.

This wisim-based tile-level flow does not support proper verification of feedthrough logic as these signals are only defined during the post layout phase of the physical design process. There is no reference point for these feedthrough input and output signals on the RTL model. For full verification coverage, these paths need to be checked. This is a particular concern when discussing possible power gating that might be along the feedthrough path. With this exception, the flow does give us confidence that the remaining logic for the tile is good. This method also assumes that verification of the tiles is confirmed if the outputs of the tile match the tile outputs in a golden RTL reference waveform. For some verification areas, where perhaps a memory dump is required to confirm a test pass, this might not be a sufficient check.

5. Background to wisim data-replay flow

This section provides a brief sampling of the input files to the Synopsys Siloti What-If-Sim-Replay to support the testbench setup, compile, and simulation. Users who are interested should get more support details from their local Synopsys representatives.

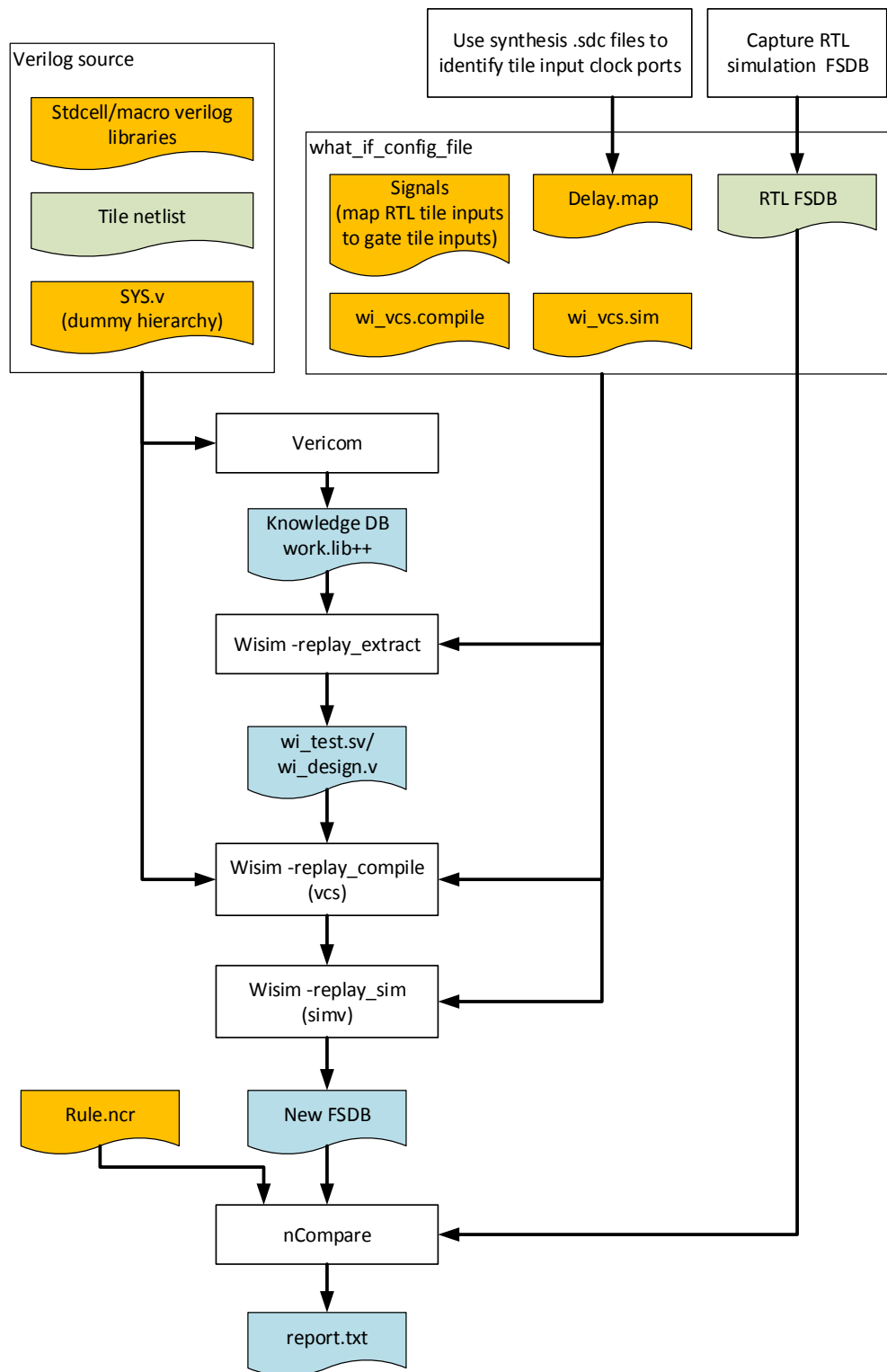


Figure 6 : Depiction of processes, files, and libraries to run 'wisim' flow

Create a file to setup your tools environment:

```
module load vcs/2014.12
module load verdi/2014.12-2
setenv NOVAS_WICG_DUMP_INITIAL 1
setenv NOVAS_LICENSE_FILE <licenseServerName>
```

Create a command to create a knowledge database:

vericom.cmd

```
vericom -lib work +libverbose -onfatalerrorcontinue -ssy -ssv -sv -vc
-wcFile +define+assertions $HOME_DIR/verilog_src/mytile1.v
$HOME_DIR/verilog_src/SYS.v -v $HOME_DIR/verilog_src/std_cells.v
```

These following files will be referenced during flow execution:

signals (map hierarchy in RTL FSDB to hierarchy to replay)

```
SYS.MPU00.gnb_top.gnb.usblk_nb.myinput1 =>
SYS.MPU00.gnb_top.gnb.usblk_nb.myinput1
```

wi_vcs.compile (Compile command:

```
vcs -full64 -sverilog -f wi_run.f -P
$NOVAS_HOME/share/PLI/VCS/LINUX64/novas.tab
$NOVAS_HOME/share/PLI/VCS/LINUX64/pli.a -debug_all -lca +define+WI_NO_DUMP
```

wi_vcs.simulate (Run command) :

```
./simv +fsdb+gate
```

SYS.v (reconstruct a dummy hierarchy above the DUT):

```
module SYS ();
    MPU00 MPU00 ();
endmodule
// SYS.MPU00.gnb_top
module MPU00 ();
    gnb_top gnb_top ();
endmodule
// SYS.MPU00.gnb_top.gnb
module gnb_top ();
    gnb gnb ();
endmodule
// SYS.MPU00.gnb_top.gnb.usblk_nb
module gnb ();
    sblk_nb usblk_nb ();
endmodule
```

what_if_config_file:

```
set FSDB = <somePath>/<goldenRTL.fsdb>
set Scope = SYS.MPU00.gnb_top.gnb.usblk_nb
set Map = <somePath>/signals
set Begin_Time = 0
set End_Time = 120000
set Time_Unit = ns
set Simulation_Compile_Script = <somePath>/wi_vcs.compile
set Simulation_Run_Script = <somePath>/wi_vcs.sim
set delay = <delay_map_file>
```

You can execute the flow using these commands:

```
source vericom.cmd
wisim -lib work -config what_if_config_file -top "SYS" -replay_extract
wisim -lib work -config what_if_config_file -top "SYS" -replay_compile
wisim -lib work -config what_if_config_file -top "SYS" -replay_sim
```

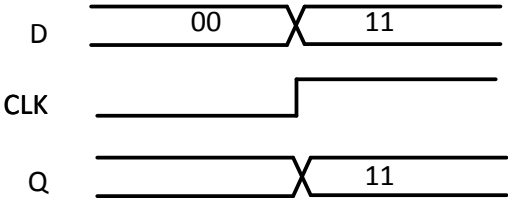
The execution of the flow will produce various source files and logs. The most relevant for our flow is the `wi_result.fsdb` which is the new FSDB output. We now use the `nCompare` tool to do a waveform comparison of the tile output signals from the RTL FSDB to the same tile output signals in the replayed FSDB. Details on using the `nCompare` waveform comparison tool are well documented in the Verdi Reference Manual. If the waveforms match we define the test as a pass result for the tile. In our AMD gate-level testcase, we observe a few design for test (DFT) output port mismatches which need to be waved as those are related to DFT implementation changes not found in the RTL code.

6. Issues to achieve ‘passing’ gate tiles through gate tile flow

Getting the flow to run is fairly straight forward. Getting the resimulated tile-level gate outputs to cycle match the RTL tile outputs was where much effort was needed. The table below lists the flow related issues which were worked through in achieving success through all the tiles in our AMD testcases.

Table 1: List of resolved issues to achieve flow success

Issues	Description
Memory prefills or forces into design	<p>With the gate tile-level replay flow, the only inputs to drive the system are the tile input ports. If the RTL environment has testbench forces into the RTL design or if there are memories that are prefilled with data through the testbench, we need to account for these in the replay flow in order to match the tile outputs to the RTL waveform reference. In these cases, it was easiest for us to simply map these additional points as points to be replayed based on the RTL reference FSDB.</p> <p>If your environment supports this, it may be preferential to configure your RTL reference environment in a way to avoid memory prefills or forces into the design.</p>
Alignment of RTL model to netlist	The flow is relying on a cycle-by-cycle functional match between the RTL model and the gate netlist representation. If the RTL reference model and the design netlist are not aligned, you could very well end up debugging functional differences that are expected.
Synchronizer cells with randomized timing	AMD utilizes some synchronizer cells with randomized cycle delays. This is to account for some uncertainty when signals go through a synchronizer from one clock domain to another. However, in this replay flow we need to disable any cycle delay randomization to ensure that the behavior of the sync cells in the gate-level tile replay directly match the behavior of the sync cells in the original RTL simulation.
Repeater flops	Repeaters flops are used along feedthrough paths to support signals

	<p>transfers that are over a long distance.</p> <p>In the AMD gate-level designs, the hierarchical position of the repeater flops varies between the RTL stage and later in physical netlist implementation. In the RTL design, the repeaters sit above the tiles whereas in the gate netlist the repeaters sit inside the tiles. To avoid this problem, we buffer out the repeaters in the gate tile netlist.</p>
Race between clock and data signals	<p>There were many cases where mismatches at a tile's output were traced all the way back to the first flop stage within the tile. The problem rooted from the fact that simply replaying RTL tile input stimulus into the gate tile netlist (even with options defined to keep event sequence ordering) made us prone to simulation race issues between clock and data signals at the first level of flops.</p>  <p>Figure 7 – Incorrect behaviour through a flop stage due to simulation race when both D and CLK change at same time</p> <p>The solution here was to simply add a small '1 ps' delay to all non-clock signals. The wisim flow supports a 'delay.map' feature which allows the user to associate a small delay to a set of signals during replay. The user can leverage the synthesis .sdc constraint files which have explicit references to identify the clock ports of the tile, and this is safer than relying on a match of "CLK" in the name of a port.</p>

7. Results

By utilizing this tile-level gatesim approach, the compile and simulation of each tile for a typical AMD graphics test can be done in a little over one hour. A regression of all tiles can be setup, and results can be obtained in a few hours depending on our access to hosts in our datacenter.

One item to note is that this replay flow has no dependency on the original RTL testbench. The focus is purely the quality of the netlist. For example, there are no testbench checkers which try to reference signals that may not exist in the netlist.

When there are failing tiles to debug, the user has immediate access to a FSDB with full visibility along with a prebuilt Verdi knowledge database (kdb) that was built to support the replay flow. There is no delay to users to begin the debug process.

8. Conclusions

This new tile-level gatesim flow provides a practical platform to support debug of most issues in the gate design. In its current state at AMD, it does not replace our SoC/Gnb gatesim effort, as there are still pieces missing for verification coverage, but it allows teams to scrub the majority of tile-level issues in an environment that allows for quick debug cycles. It also allows gate simulation teams to be free from working within the access limitations of only a few large memory hosts.

There are some additional side benefits that come with using this flow:

- It is a true parallelized tile-level debug. We are feeding healthy passing stimulus to each tile instance, and we have a reference to passing output values. Each tile is simulating within its own independent testbench. We are no longer onion peeling at the SoC/Gnb-level, where an issue in one tile needs to be fixed before we can simulate further to expose another issue in a different tile.
- Debug of gate tiles can begin as individual tiles come out of the physical design team. You do not need to stall your gatesim effort until all tiles in the design are ready.
- Being able to replay a waveform also implies that we could use a ‘failing’ waveform from the SoC gatesim environment which has limited debug visibility. If the SoC FSDB includes the tile top-level ports, you can use it in this tile-level gate regression to replay and recreate full visibility waveforms for all the tiles in the design within a reasonable time.
- Whereas some verification flows in the past might have been restricted due to simulation size or runtime, this flow can provide a renewed option for things like running SDF full timing gate regressions or creating long gate-level vectors for power estimation.

9. Acknowledgment and References

I would like to thank Myles Glisson and Ravi Gehani of Synopsys for their assistance in the writing of this paper and corresponding presentation slides.

I would like to thank Robert Matyjewicz for all his effort during his AMD internship in helping to bring up this flow and in supporting its deployment.

AMD, the AMD Arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.