Method to Partition Your Gate Simulation Debug

Kendall Chan

AMD, Markham

October 1, 2015

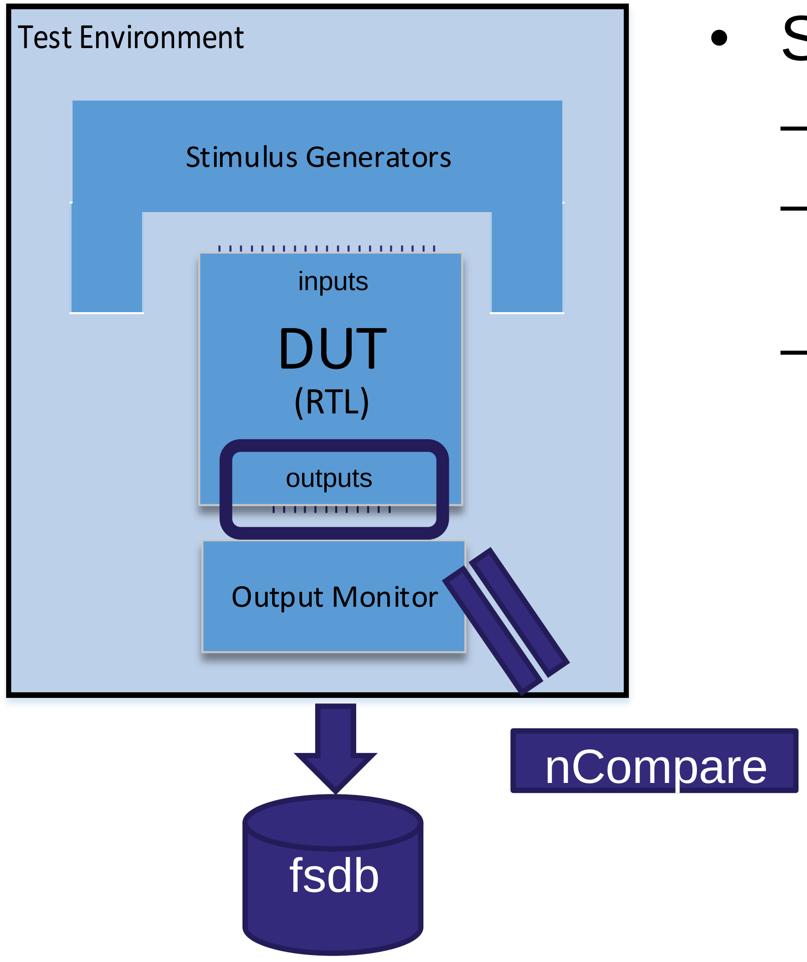
Ottawa, Ontario, Canada

Agenda

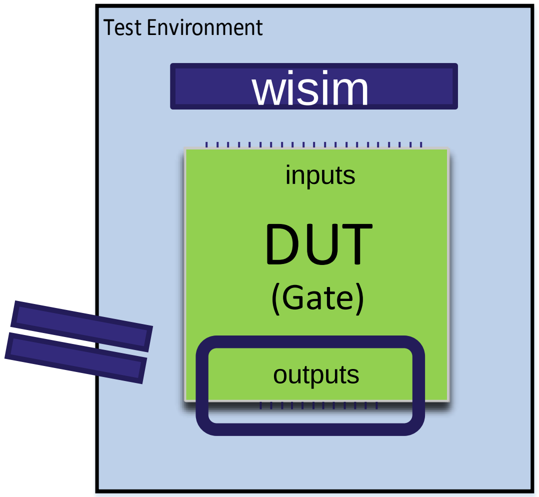
Overview

- AMD's gate simulation dilemma
  - Multi billion gate designs
  - Requires 256 Gb memory hosts
  - Why run?
    - Good final check on power logic inserted during physical design flow
    - Formal tools don't seem to provide reliable end to end check for RTL + upf through to post layout power aware netlist
    - We still find the rare bug

Overview



- Siloti What-if Sim Data Replay
  - Abbreviate to be ‘wisim’
  - Replay fsdb from specific simulation time and run till end time
  - Can apply changes into env to evaluate effect - ‘what if’?



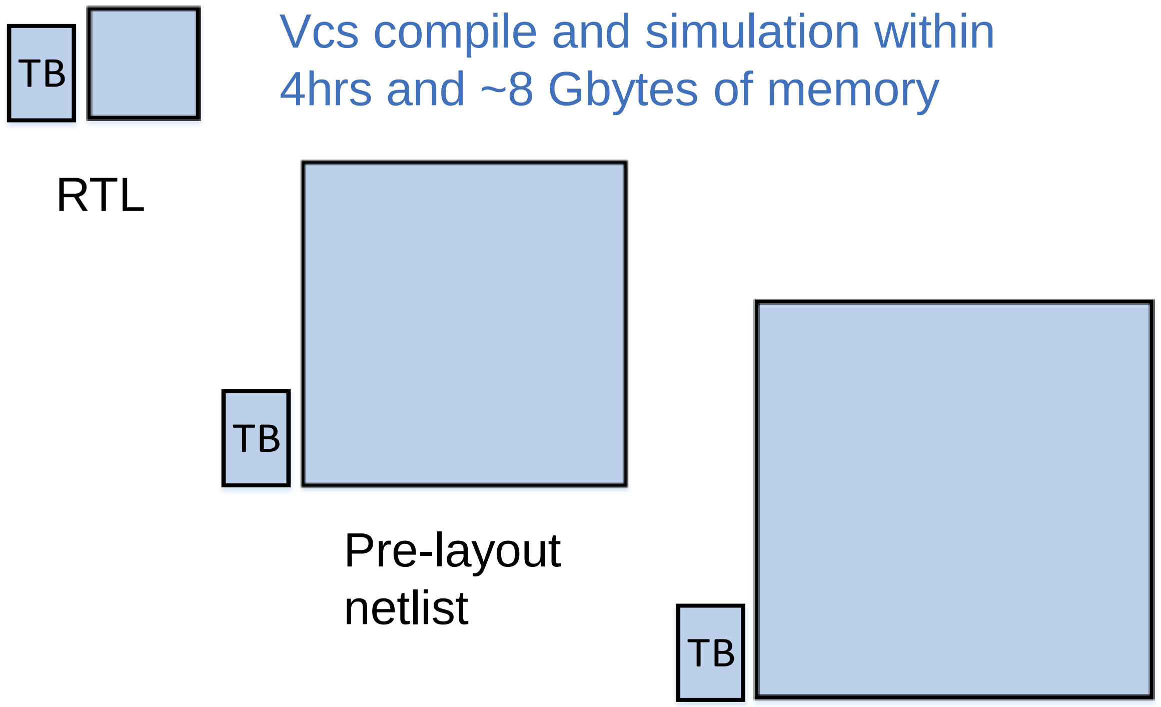
Question??

How to define expected behaviour?

How to compile?

Gate Simulation Debug Challenges

1. Slow compiles, long simulation times, high memory



Vcs compile and simulation within 4hrs and ~8 Gbytes of memory

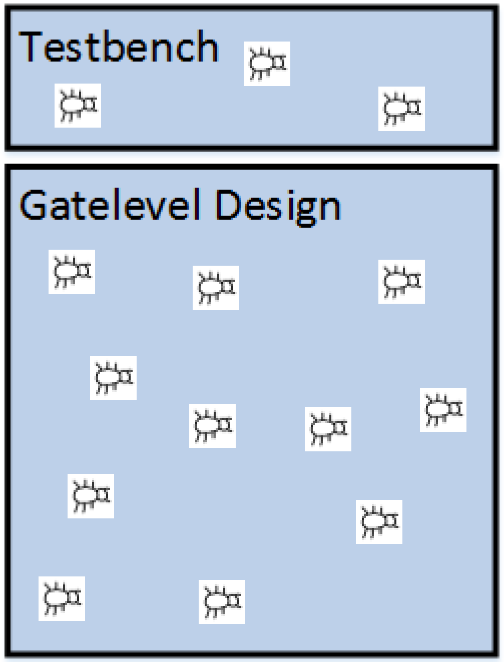
VCS compile ~2 days
Simulation time 2+ days
110+ Gbytes of memory

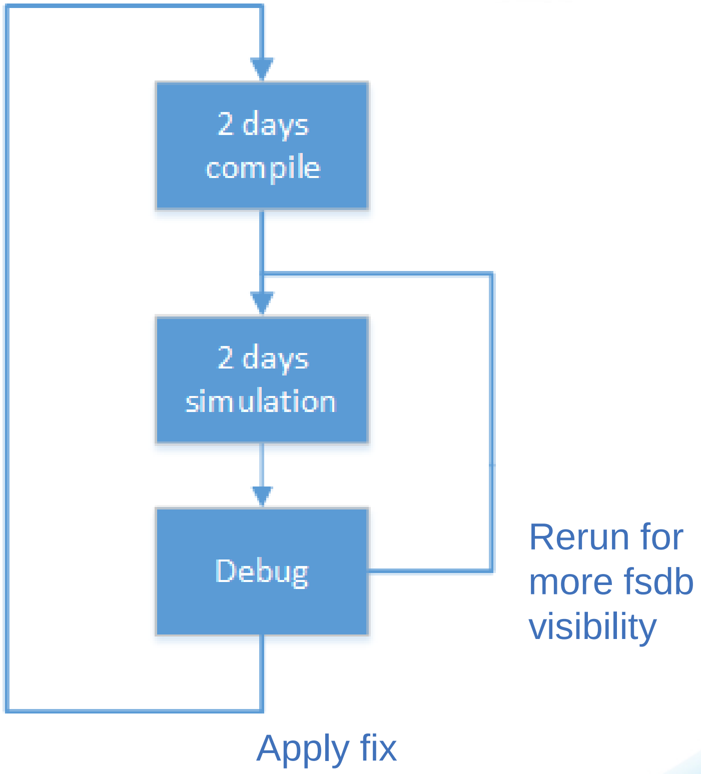
Gate Simulation Debug Challenges (continued)

2. Testbench issues
   - Programming sequence
   - Logic for ‘forces’ or prefilling of cache memories
   - Checkers/Monitors
3. Netlist and library quality
   - Verification team needs to ramp effort ahead of physical team completing all their checks
   - Technology stdcell/macro library may have untested modeling problems

Gate Simulation Debug Challenges (continued)

4. Long iterative debug process
   - "onion peeling" – find & fix first bug, then repeat

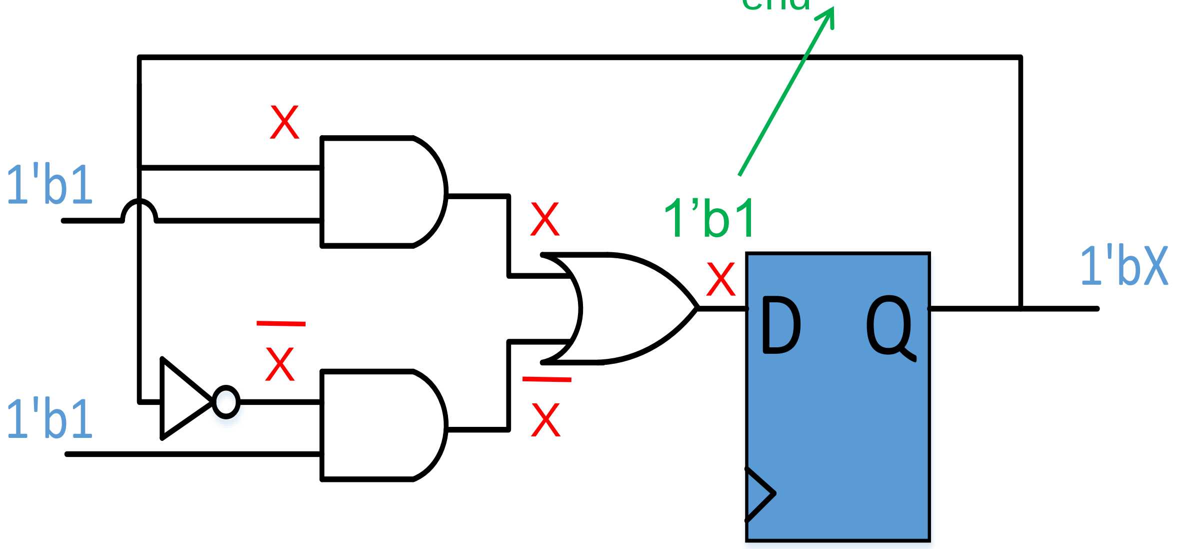




Gate Simulation Debug Challenges
(continued)

5. Need to apply $deposits

```
Initial begin
 $deposit (<somehierarchy>.Q, 1'b1);
end
```



6. Stuck waiting until entire gate netlist is available

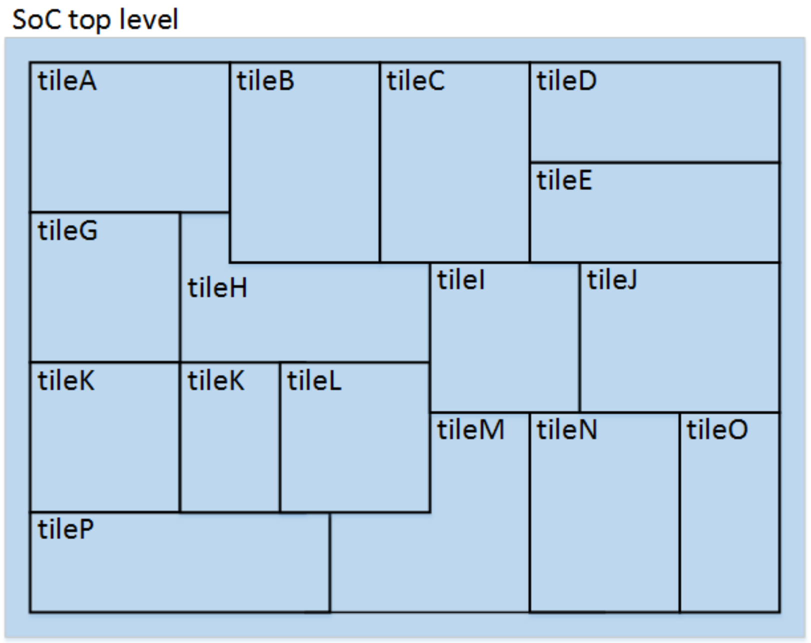
Past Strategies

- Options most teams consider:
  - Find large and fast dedicated machines
  - Mix RTL and Gate blocks
  - Siloti
    - Increase debug visibility
      or
    - Reduce runtime for same visibility
  - Do we really need to run gate simulations?

Agenda

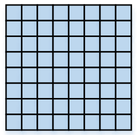
Introduction to Tiling

- Large SoC ASIC designs typically use a tiled-based approach to segment the physical floorplanning effort



- These ‘tile’ layers are preserved from RTL through netlist implementation

Introduction to Tiling



RTL

(1 VCS process)

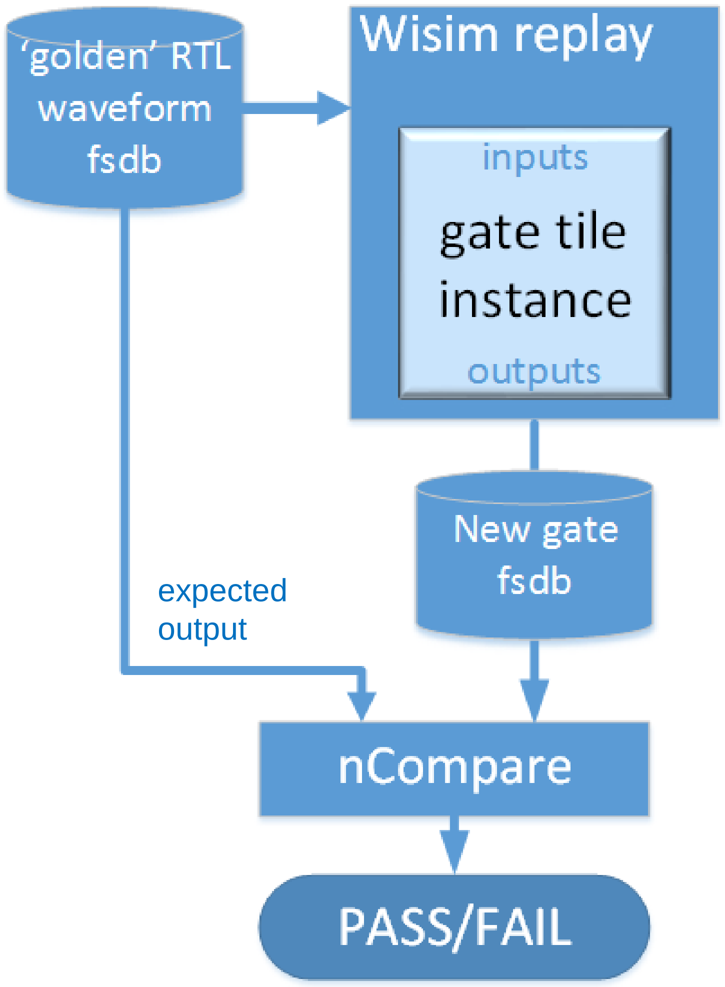
Compile+Run
~4 hr

Compile+Run
~4 days

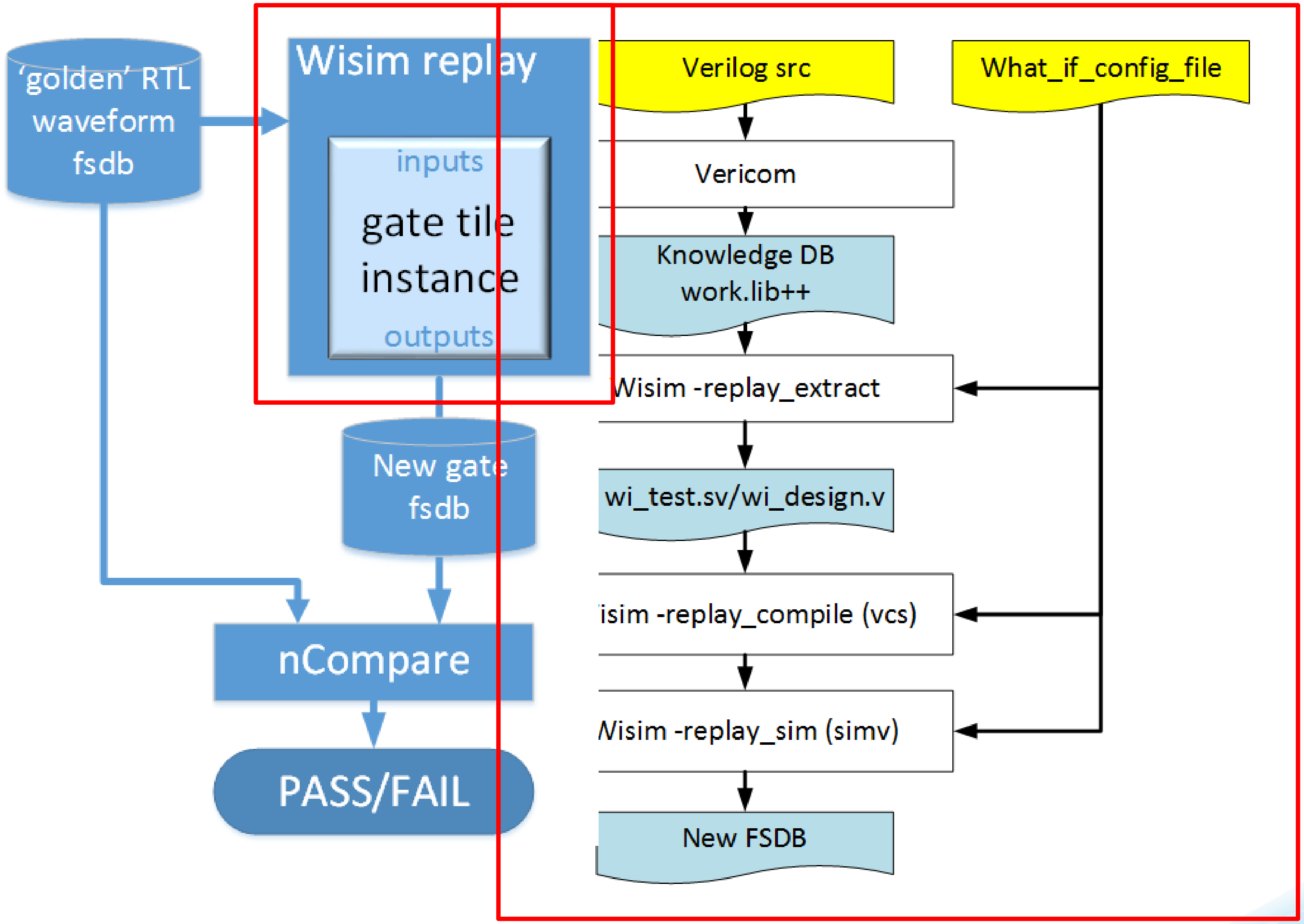
Compile+Run
~1.5 hrs/tile

Siloti What-If Data Replay Flow

- Key flow execution notes
  - VCS is controlled by Wisim engine
  - Applies FSDB to gate tile instance
  - Output is a full gate tile FSDB
  - In-house regression scripts to parallelize setup and execution



Siloti What-If Data Replay Flow



What-if Data Replay Setup Files

Sample files referenced during flow execution:

what_if_config_file:

```
set FSDB = <somePath>/result.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 = <somePath>/delay.map
```

What-if Data Replay Setup Files

Sample files 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
```

delay.map:

```
SYS.MPU00.gnb_top.gnb.usblk_nb.FOO #1000
```

What-if Data Replay Setup Files

Sample files referenced during flow execution:

SYS.v (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 Data Replay Setup Files

Setup file to load your tools:

```
module load vcs/2014.12
module load verdi/2014.12-2
setenv NOVAS_WICG_DUMP_INITIAL 1
```

vericom.cmd to create a knowledge database:

```
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
```

WISIM commands (4 Steps)

- Knowledge Database compile

```
source vericom.cmd
```

- Wisim Extraction

```
wisim -lib work -config what_if_config_file -top
"SYS" -replay_extract
```

- Wisim Compile

```
wisim -lib work -config what_if_config_file -top
"SYS" -replay_compile
```

- Wsim Simulation

```
wisim -lib work -config what_if_config_file -top
"SYS" -replay_sim
```

nCompare FSDB Comparison Utility

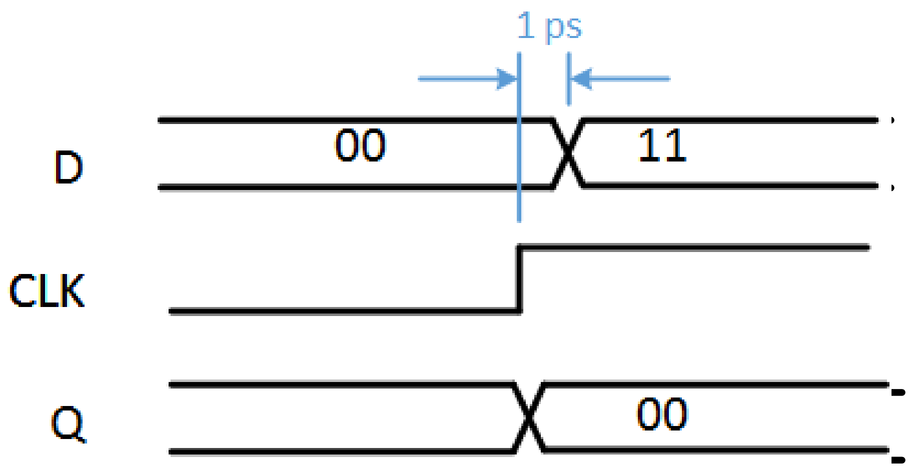
- Pass/Fail? - Use nCompare to compare tile outputs of original RTL FSDB against replayed gate tile FSDB
- Please refer to nCompare documentation for further info

Sample lines from a nCompare report.txt:

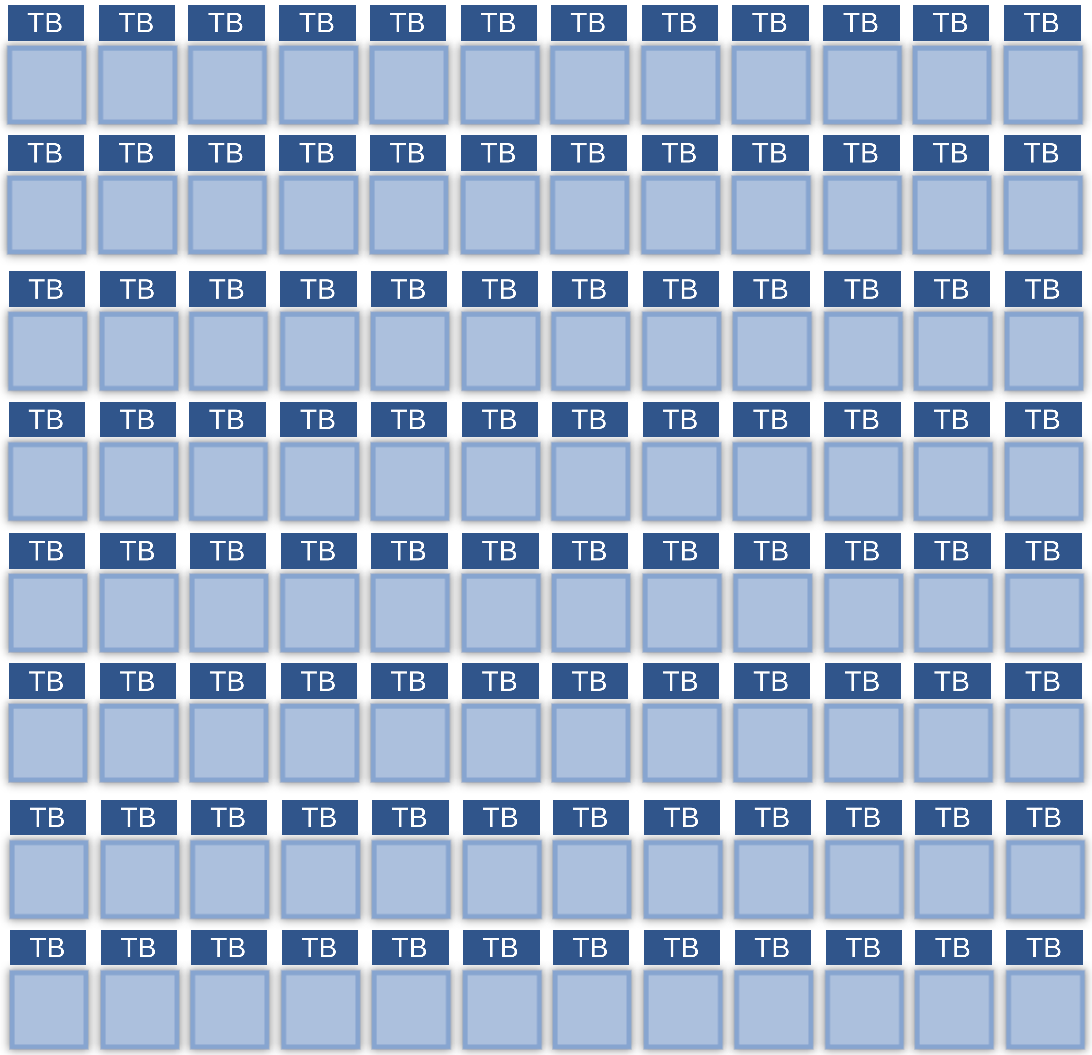
```
510 9475000000fs 9475000000fs 500000fs 500000fs
`/SYS/MPU00/gnb_top/gnb/usblk_gck/CK_GPU_ACLK`
`/SYS/MPU00/gnb_top/gnb/usblk_gck/CK_GPU_ACLK` 3 1 2 1 0
540 9475000000fs 9475000000fs
`/SYS/MPU00/gnb_top/gnb/usblk_gck/CK_GPU_ACLK`
`/SYS/MPU00/gnb_top/gnb/usblk_gck/CK_GPU_ACLK` 3 1 2
510 9480000000fs 9480000000fs 5000000fs 5000000fs
`/SYS/MPU00/gnb_top/gnb/usblk_gck/GN_GDDR_ClockIn_H`
`/SYS/MPU00/gnb_top/gnb/usblk_gck/GN_GDDR_ClockIn_H` 3 1 2 X 1
```

Issues Resolved to Achieve Success

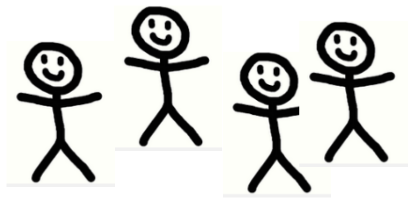
- Memory prefills or forces into design
- Alignment of RTL model to netlist
- Synchronizer cells with randomized timing
- Repeater flops
- Race between clock and data signals



Regression


TB TB TB TB TB TB TB TB TB TB TB TB
TB TB TB TB TB TB TB TB TB TB TB TB
TB TB TB TB TB TB TB TB TB TB TB TB
TB TB TB TB TB TB TB TB TB TB TB TB
TB TB TB TB TB TB TB TB TB TB TB TB
TB TB TB TB TB TB TB TB TB TB TB TB
TB TB TB TB TB TB TB TB TB TB TB TB
TB TB TB TB TB TB TB TB TB TB TB TB

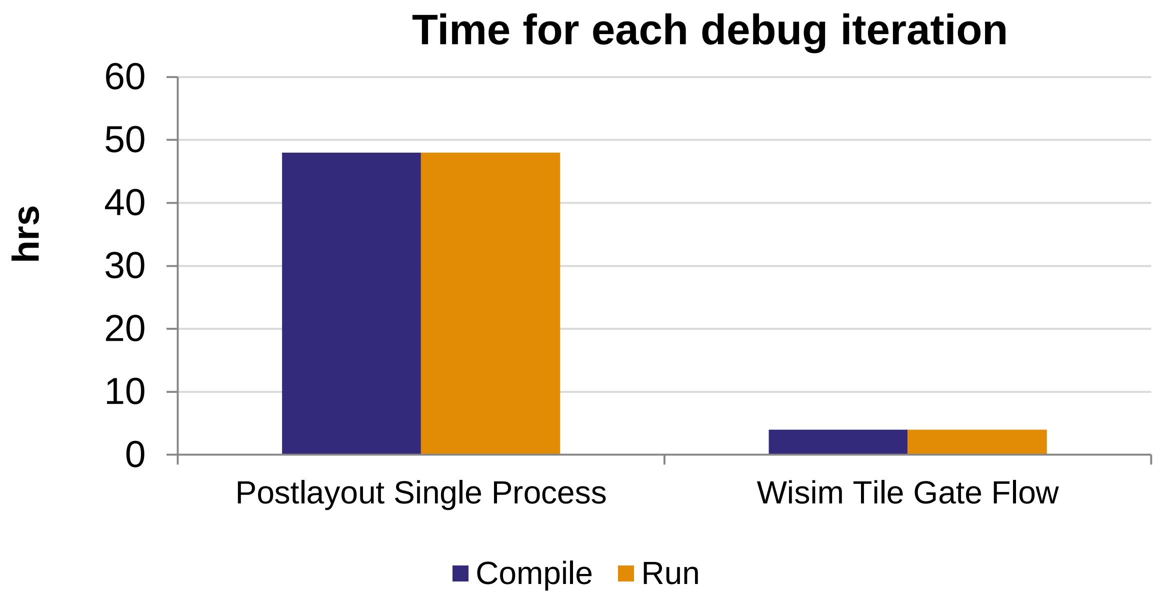
- First - run all tiles without $deposits
- Next - rerun failing tiles with $deposits
- Assign remaining failures for debug



- X-prop
- Modeling issues/ RTL artifacts/ Netlist bugs

Results

- Compile and run of GFX test in ~1-2 hr/tile
- Run a regression of tiles together on LSF farm
- No debug delay - immediate access to full FSDB and Verdi knowledge DB (kdb) for failing tiles
- No dependency to fixing testbench issues to match netlist



Conclusions

Gate Debug Challenges	Tile Level Gate Result	
Slow compiles, long simulation times, high memory	Compile/run < 2 hrs/tile, ~2 Gbyte memory	✓ $$$
Testbench Issues	Avoids programming issues and no testbench monitors & checkers. Need to still fix forces/prefilled memories	✓
Netlist and library quality	Localizes debug to tile level	✓
Onion peeling for debug	Can directly attack bugs across all tiles	✓
Need for $deposits	Many tiles pass without any deposits	✓
Stuck waiting until entire gate netlist is available	Can begin as first tile netlists are available	✓

Questions?

Any questions?

AMD

snug
Synopsys Users Group

Thank You

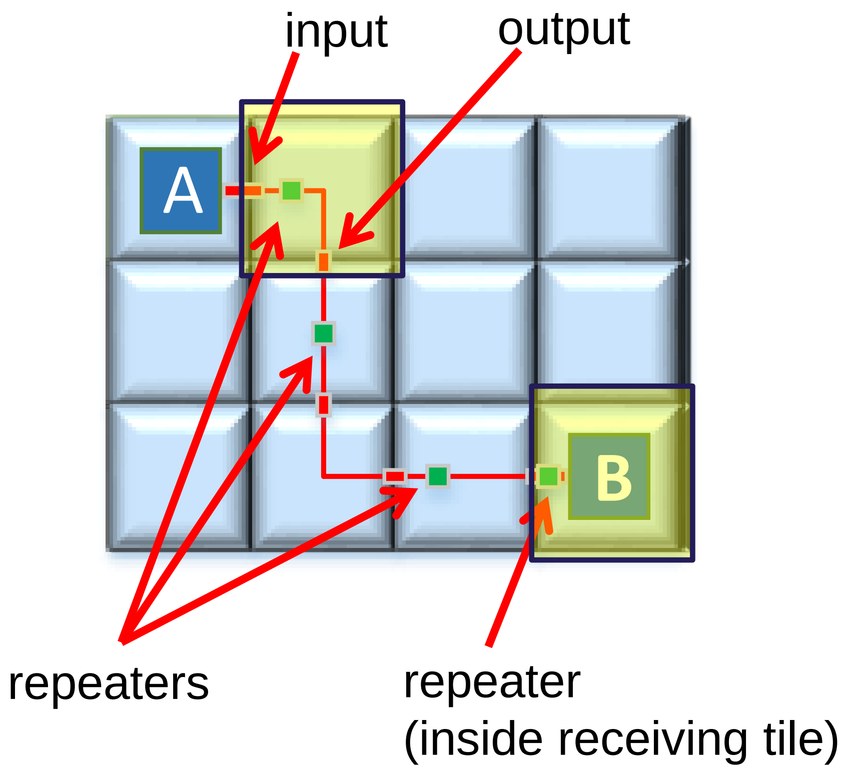
Backup Slides

Next Steps

- Compare all registers in tiles – not just tile outputs
- Deploy Siloti to only dump essential signals to minimize fsdb dump size
- Revive your SDF efforts or your gate-level power estimation efforts

Tile Gate Simulation Flow

- What about feedthroughs?
  - Not part of RTL - inserted during physical design floorplanning stage
  - No effect on logic internal to tiles
  - Conclusion : Ignore feedthroughs
- What about repeaters?
  - AMD repeaters sit in different hierarchies between RTL and netlist
  - Conclusion: Must remove repeater flops



Disclaimer & Attribution