

Advanced X-Prop Usage for the NXP LS1088a Verification

Jie Wen – NXP Semiconductors

Jiri Prevratil – Synopsys

Amol Bhinge – NXP Semiconductors

Vaibhav Kumar – NXP Semiconductors

Sept, 2016

Austin



Agenda

Why X-Prop?

Simulation and debugging with X-Prop

Results of X-Prop Deployment

Future Work and Enhancements

Conclusions

Why X-prop?



Why X-Prop?

Typical X Propagation Problem

Symptom: RTL simulation yields 0 or 1 on a net,
but Gate simulation yields X on the same net

Cause: Two different mechanisms

- X-Optimism in RTL simulation
- X-Pessimism in gate simulation

Severity:

- RTL X-Optimism: potential false pass
 - Functional bug concealed in RTL
 - May be caught by gate simulation, if you run the right stimulus
- Gate X-Pessimism: potential false failure
 - Limitation of the simulation model, X may not be true design bug
 - Debugging mismatch at gate level is very expensive

Why X-Prop?

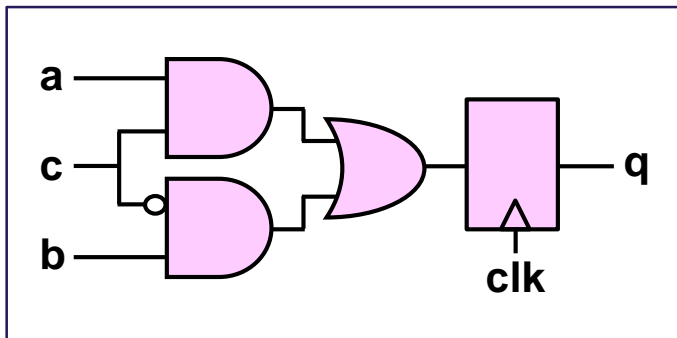
Are You an Optimist or Pessimist?

```
process (clk) begin
  if (c = '1') then
    q <= a;
  else
    q <= b;
  end if;
end process;
```

VHDL

```
always @(posedge clk)
  if (c)
    q <= a;
  else
    q <= b;
```

Vlog



Optimistic RTL

c = X
a = 1
b = 0

RTL: q = 0

Gate: q = X

Actual: q = ?

Pessimistic Gate

c = X
a = 1
b = 1

RTL: q = 1

Gate: q = X

Actual q = 1

Solution: VCS X-Prop

Accurate (but non-standard) semantics for RTL simulation

- **Approximate don't-care semantics in RTL simulation**
 - Consider effect of 0 / 1 for every X-controlled assignment or event
 - Reduce RTL-synthesis/gate mismatches
 - Keep overhead within reasonable bounds
- **Built-in mechanism to control pessimism/optimism**
 - Pessimism is more conservative: safer → desirable
 - Avoid overly pessimistic semantics, particularly earlier in project

What is happening under the hood?

X-Prop Merge Modes

- VCS X-Prop simulator considers the effect of both 0 and 1 for every X
- T-merge and X-merge mode setting determines the level of pessimism

- **T-merge**: Yields X when values are different
 - Recommended for first pass to find X-related bugs
- **X-merge**: Yields X always (more pessimistic)
 - Never more optimistic than any “gate” implementation

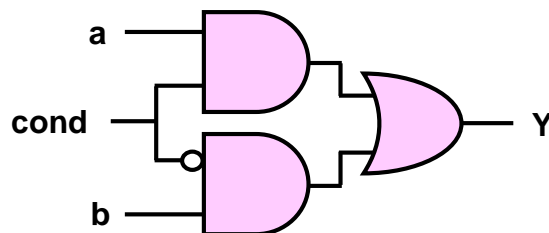
Standard RTL hides a bug

Gate X, unnecessary debug

X-merge is most conservative

```
always @(posedge clk)
```

```
if( cond )  
    Y = a;  
else  
    Y = b; Vlog
```



cond	a	b	Std RTL	RTL T-merge	RTL X-merge	Gate Y
X	0	0	0	0	X	0
X	0	1	1	X	X	X
X	1	0	0	X	X	X
X	1	1	1	1	X	X

References: VCS User Guide, SolvNet articles,
SNUG Austin 2013: [A Practical Approach to Implementing Synopsys X-Prop Technology on a Complex SoC](#)

Simulation and debugging with VCS X-Prop



Simulation and Debugging with X-Prop

Compiling for X-Propagation

- Without configuration file
 - All code is instrumented using T-merge (default)

```
vcs -xprop test.v
```

- With configuration file (recommended)
 - Instrumentation is applied to specific instances/hierarchies/modules
 - “Bottom-up” flow is recommended
 - Apply to synthesizable code, not testbench code

```
vcs -xprop=xprop.cfg test.v
```

Simulation and Debugging with X-Prop

LS1088a Configuration

- Disable testbench coded from the top
- Enable X-Prop on DUT
- Disable specific instances
- Disable specific modules

xprop.cfg

```
//Disable Xprop for entire testbench tree
tree { testbench } { xpropOff };
//Enable Xprop for entire DUT
instance { testbench.top_ls1080 } { xpropOn } ;

//----- EXCLUDE based on VC Static Report -----
//instance { testbench.top_ls1080.aiop_cplx } { xpropOff } ;
instance { testbench.top_ls1080.qbman_cplx } { xpropOff };
instance { testbench.top_ls1080.a53_c1 } { xpropOff };

//this is for rgmii test which might have race condition
//module { rgmii_conv } { xpropOff } ;

module { DWC_usb3_GTECH_FD2 } { xpropOff } ;
module { DWC_usb3_GTECH_FD4 } { xpropOff } ;
module { DWC_usb3_GTECH_LD2 } { xpropOff } ;
```

Best Practice:


- Disable X-Prop on all non-synthesizable modules
- Synopsys VC Static is used to extract non-Synthesizable modules
- “Bottom-up” flow is recommended; Focus on interesting blocks first then move up to SoC

References: SNUG Austin 2013: [A Practical Approach to Implementing Synopsys X-Prop Technology on a Complex SoC](#)

Simulation and Debugging with X-Prop

X at Reset

- Flops with a pipelined asynchronous reset are in an unknown state in the first a few cycles of simulation
- When X-prop is turned on, these known Xs at time 0 and the first a few cycles before the reset raised a false alarm
- VCS provides the task `$set_x_prop` for user to turn on/off the X-prop semantics dynamically



```
`ifndef GATE_MODEL
`ifdef VCS
initial
begin
    $set_x_prop("vmerge"); // vmerge is default Verilog behavior
    repeat (32) @(posedge `SOC_DUT_TOP.SYSCLK);
    $set_x_prop("tmerge"); // normal xprop behavior
    @(`negedge testbench.tbw_PORESET_B);
end
`else
.....
`endif
`endif //`ifndef GATE_MODEL
```

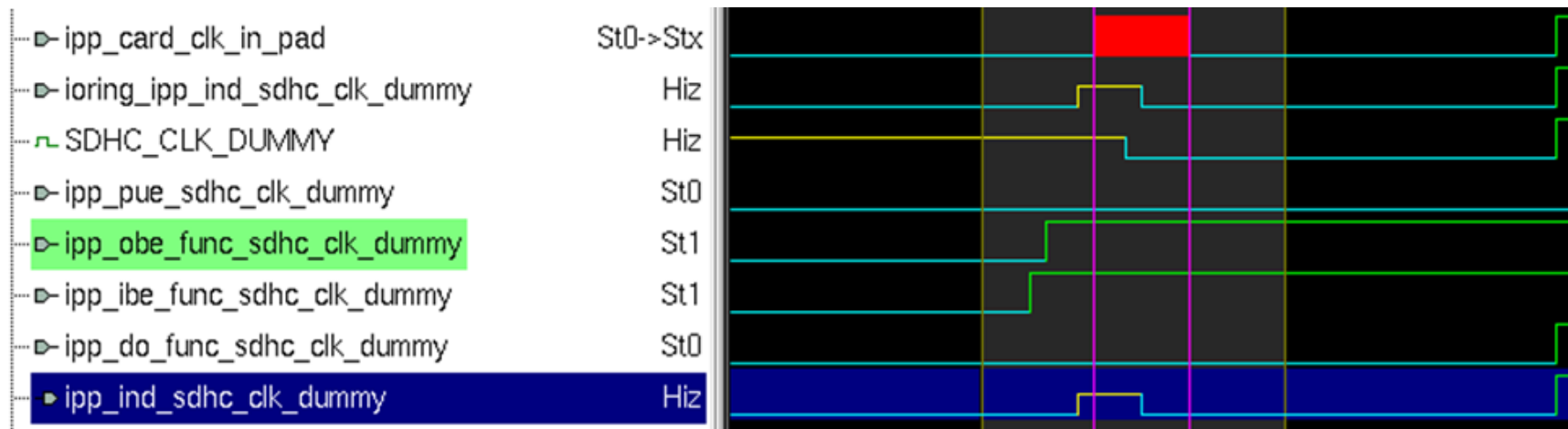
Best Practice:

- Turned off X-Prop at time 0 and first a few cycles to disable the false alarm and detect the potential 'X' issue after time 0

Simulation and Debugging with X-Prop

X-Prop Debug with Delta Cycle

- Delta Cycle feature of the gui is used to see the glitch within a timestep in simulation
- To Enable Delta Cycle feature
 - Using system task \$vcdplusdeltacycleon or UCLI command “dump deltaCycle on” for VPD
 - Using simulation command-line options “+fsdb+glitch=0 +fsdb+sequential” for FSDB
- To View Delta Cycle feature
 - Via the right-click menu selection “Expand Time” in DVE
 - Via the “View | Expand Delta” menu selection in Verdi



Simulation and Debugging with X-Prop

X-Prop Debug with Verdi



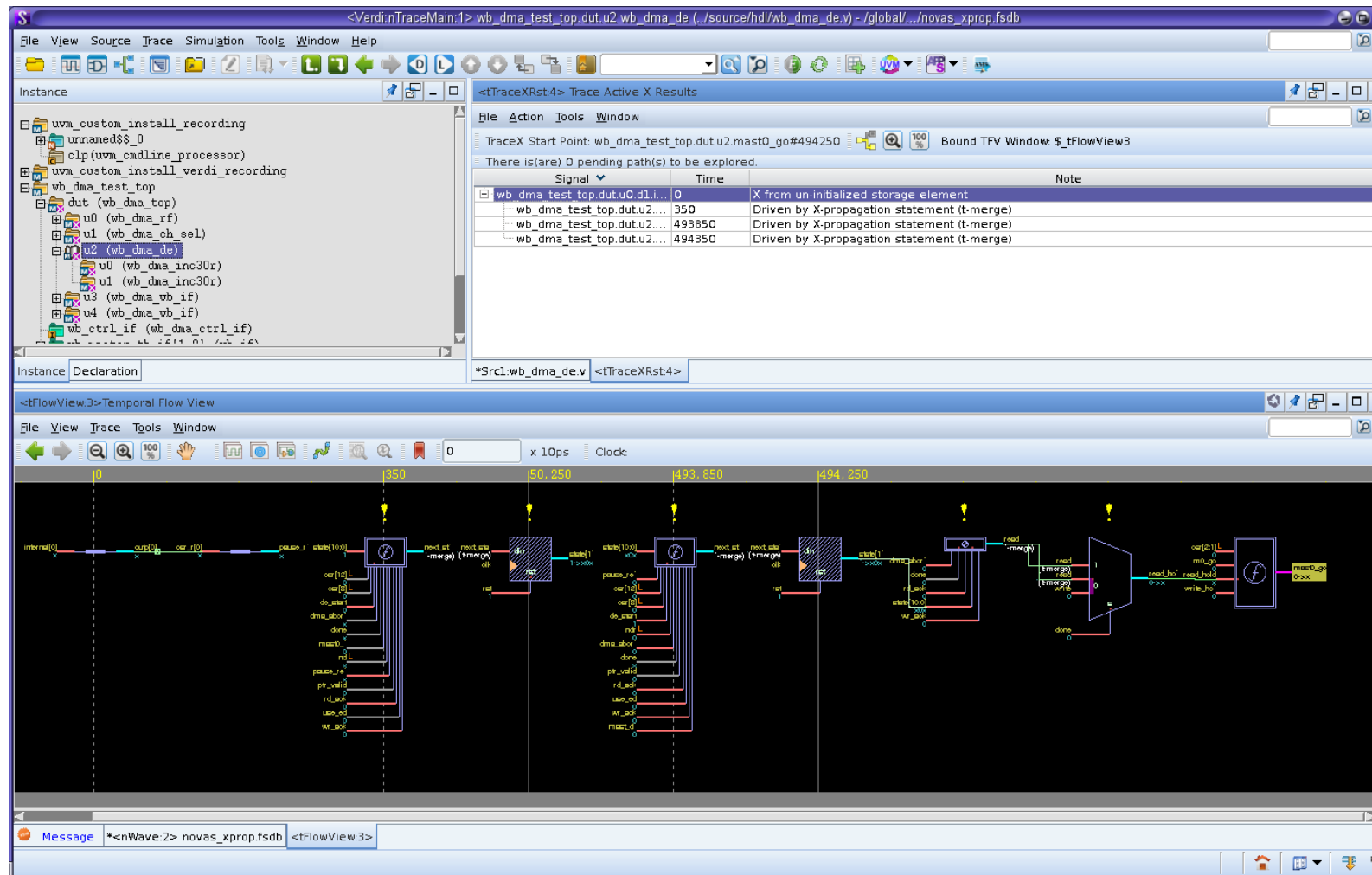
- Debugging simulations with X-Prop mainly uses normal X tracing methods
 - Driver tracing
 - “Active X” tracing
 - Verdi Temporal Flow View (TFV)
- Verdi provides additional X-Prop assistance
 - Source code coloring indicates instrumented code
 - Link from VCS instrumentation log to source code
 - TFV “t-merge” and “x-merge” indicators
 - Driver tracing list result indicates “TDT” or “TDX” for merge type

```
verdi -f run.f -ssf myfile.fsdb \  
-xprop xprop.cfg -xproplog xprop.log
```


Simulation and Debugging with X-Prop

X-Prop Debug with Verdi

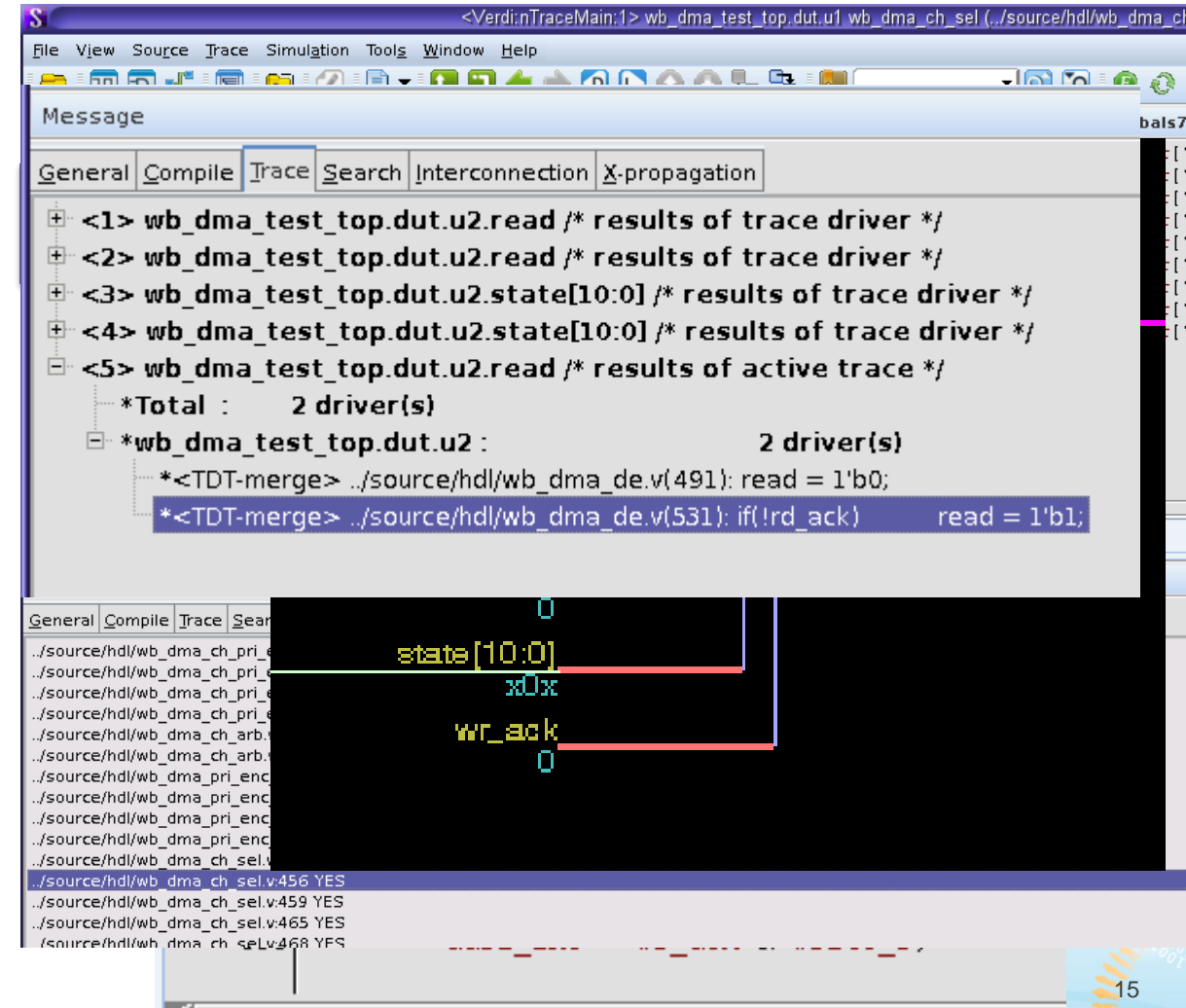
- Verdi Temporal Flow View - Trace X results



Simulation and Debugging with X-Prop

X-Prop Debug with Verdi

- Debugging simulations with X-Prop mainly uses normal X tracing methods
 - Driver tracing
 - “Active X” tracing
 - Verdi Temporal Flow View (TFV)
- Verdi provides additional X-Prop assistance
 - Source code coloring indicates instrumented code
 - Link from VCS instrumentation log to source
 - TFV “t-merge” and “x-merge” indicators
 - Driver tracing list result indicates “TDT” or “TDX” for merge type



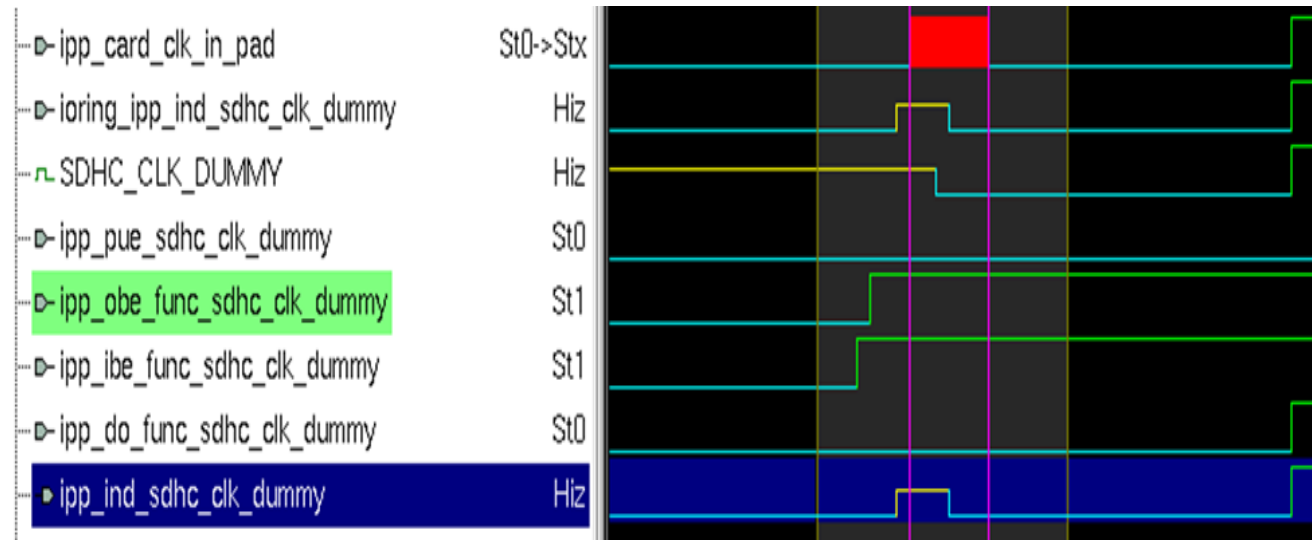
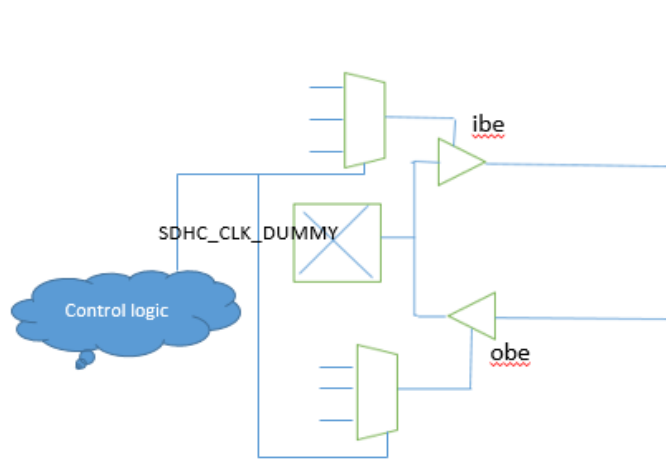
Results of X-Prop Deployment



X-Prop Deployment on LS1088a

Undriven Pin and Race Condition (RTL Bug)

- This test fails due to the X in data fifo.
- The glitch on ipp_card_clk_in_pad is observed
- The race of simultaneous switching of signals causes Xs



X-Prop Deployment on LS1088a

Undriven Pin and Race Condition (RTL Bug)

- In this design, IBE and OBE are tied to 1'b1 and the tie constant values do not propagate until RCW is loaded to select eSDHC interface
- Possible race condition on IBE/OBE which could cause a glitch on SDHC_CLK_DUMMY if IBE wins the race over OBE
- This test failure uncovered 2 design issues:
 - SDHC_CLK_DUMMY pin is not driven
 - Race condition between ibe and obe in iomux logic
- Solution:
 - Testbench adds weak pull down on io_sdhc_clk_dummy pad as a workaround for RTL simulation
 - The race condition was addressed at gate netlist by addressing timing of ibe/obe correctly

X-Prop Deployment on LS1088a

Race Condition on MUX

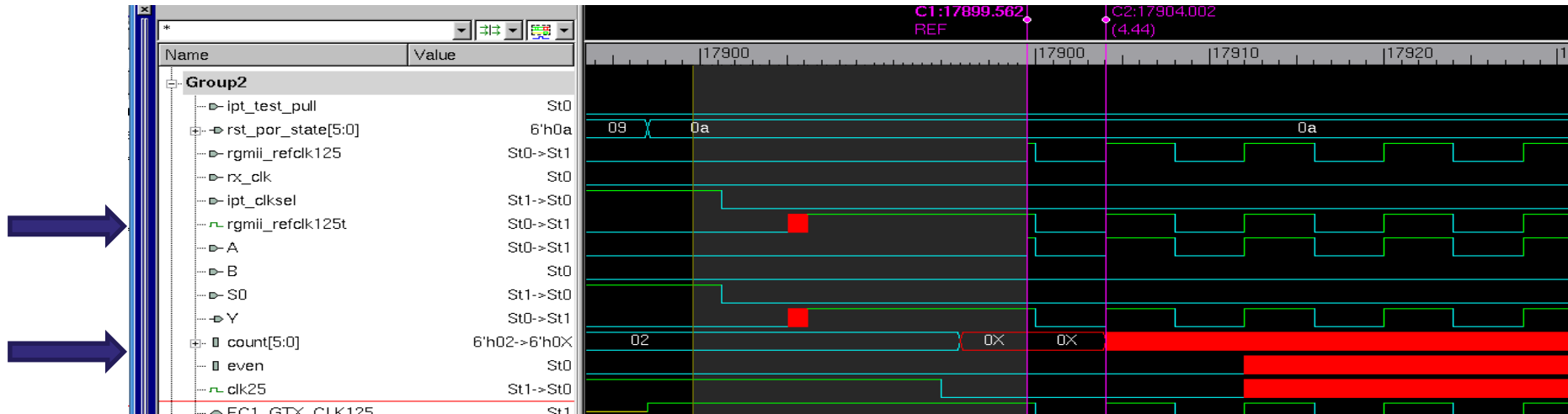
- The X is generated on rgmii_refclk125t when MUX switches
- This 'X' triggered the always block as 0->X posedge in normal (Verilog) RTL simulation
 - count increments
- X-Prop merges results of triggering and not triggering always block
 - count value is X

```
ipd_bt_2p_mux_macro ipdclkmux1 (  
    .data0          ( rgmii_refclk125 ),  
    .data1          ( rx_clk ),  
    .sel            ( ipt_clkssel ),  
    .data_out       ( rgmii_refclk125t ),  
    .ipt_test_pull  ( 1'b0 )  
)
```

```
always@ (posedge rgmii_refclk125t or posedge reset_tx_clk)  
if (reset_tx_clk)  
begin  
    count <= 6'h00;  
    ....  
end  
else  
begin  
    if (oldsel125)  
    begin  
        count <= 6'h00;  
    ....  
    end  
    if (oldsel25)  
    begin  
        count <= (count==6'd4) ? 6'h00 : count + 6'h01;  
        ....  
    end  
    if (~oldsel125&~oldsel25)  
    begin  
        count <= (count==6'd49) ? 6'h00 : count + 6'h01;  
        ....  
    end  
end  
end
```

X-Prop Deployment on LS1088a

Race Condition on MUX



- Effects of the X were thoroughly analyzed with the design team
- The `count` value affected the pulse width of a divided clock
 - The resulting potentially shorter clock width during the initial startup of that clock was determined to be harmless
- The same race condition and X is also observed at gate level simulation
 - It was much easier to debug in RTL than in gate

X-Prop Deployment on LS1088a

DMA Scenario

```
always @( posedge Sys_Clk or negedge Sys_Clk_RstN )
    if ( ! Sys_Clk_RstN )
        Gnt <= #0.001 ( 40'b0 );
    else if ( ( ( ReqArbIn | Gnt ) ) )
        Gnt <= #0.001 ( u_869 & ~ { 40 { Update } } );
    assign Sys_Pwr_Idle = 1'b1;
```

- In this scenario, signal `ReqArbIn` signal has indeterminate value, so `Gnt` in above code gets X in X-Prop simulation
- This X propagates to DMA signal `axi_bresp` which causes the corruption of data inside DMA

```
// Write response WTE set
assign data_rsp_fifo_bdone_wte_set = (axi_bresp[ 1: 0] != AXI_RSP_OKAY[ 1: 0]);
```

X-Prop Deployment on LS1088a

DMA Scenario

- `axi_bresp` was used directly to enable writing to the register holding the system bus error signals, but could be indeterminate value ('X') as described previously
- The valid signal, `axi_bvalid`, should be used to qualify `axi_bresp` to prevent the 'X' on `axi_bresp` propagating to signal `data_rsp_fifo_bdone_wte_set`

```
// Write response WTE set
assign data_rsp_fifo_bdone_wte_set =
    axi_bvalid &
    (axi_bresp[ 1: 0] != AXI_RSP_OKAY[ 1: 0]);
```

- In normal RTL simulation, the 'X' on `ReqArbIn` and `Gnt` did not propagate to `axi_bresp`, resulting in simulation false PASS
- VCS X-Prop provided the mechanism where signals with 'X' were not hidden by RTL code, and could eventually reveal the design defect

Future Work and Enhancement



Future Work and Enhancement

- Non-resettable flip-flops are a known source of 'X' origination, and cause false alarms when using X-Prop
 - Expand VC Static usage to extract non-resettable flip-flops from design, so they can be initialized to remove these false alarms
- Delta Cycle mode in the GUI helps to see glitches, but in the MUX race case, it could not provide enough information to determine the root cause of the 'X'
 - Further investigation is required to determine cause of the 'X'
- Any automation, quick trace mechanism and tool intelligence will be useful to allocate the bug quickly in this area
- Completion metric of X-Prop is desired to measure the progress
- One Step solution of X-Prop simulation
 - Detect non-synthesizable modules and disable X-Prop on those modules
 - Detect and initialize non-resettable modules with random values

Conclusion



Summary of Best Practice on X-Prop

- Use X-Prop configuration file to exclude non-synthesizable modules
- “Bottom-up” approach is recommended for complicated SoC
- Disable X-Prop at time 0 and first a few cycles to remove the false alarms and detect potential ‘X’ issue after time 0
- Use Delta Cycle to uncover glitches and the source of ‘X’
- Use ‘trace X’ and Verdi Temporal Flow View for X tracing

Conclusion

- X-Prop successfully detected bugs in RTL which would have gone undetected
 - X-Prop was applied on PASSED regular RTL regressions and some tests FAILED
 - X-prop uncovered logic issue which could be found only by gate level simulations
- X-prop provides the mechanism where inputs could be driven as 'X' which eventually reveal the design defect
- 3 logic bugs are found by running 40% of the regressions on a very stable logic
- Even with advanced debug tools like Verdi, tracing 'X' bugs remains difficult

Q & A



Thank You

