Advanced X-Prop Usage for the NXP LS1080 Verification

Jie Wen – NXP Semiconductor

Jiri Prevratil – Synopsys

Amol Bhinge – NXP Semiconductor

Vaibhav Kumar – NXP Semiconductor

NXP Semiconductor

Austin, Texas

www.nxp.com

ABSTRACT

Complex SoC design and verification environments have some signals with indeterminate value which will be represented as 'X' in simulation. The 'X' might be introduced because of different reasons, such as un-initialized signals, multiple drivers of a shared signal, timing violations on the flop, etc. The uncertainty of 'X' value might affect the functional correctness of the design. The current RTL simulation is 'X' optimistic and gate simulation is overly pessimistic and resource hungry. Synopsys X-Prop technology models Boolean uncertainty more accurately at the RTL level. It helps to identify potential design defects. This paper will discuss the advance usage of X-prop on a real complex SoC design such as NXP LS1080. It will demonstrate X-prop configuration setup for Synopsys VCS, non-synthesizable module and non-resettable flop extraction with Synopsys VC-Static and defect analysis with Synopsys Delta Cycle.

Table of Contents

Table of Figures

1. Introduction

Register Transfer Level (RTL) Simulation is a common technique used for functional verification to ensure the design meets the functional specification. A commonly recognized limitation of traditional RTL simulation is the 'X' optimistic simulation semantic which does not represent indeterminate values accurately in logic design. Gate level simulations are intended to be more realistic on the uncertainty of indeterminate value 'X', but are overly pessimistic and resource hungry. Synopsys X-prop technology models Boolean uncertainty more accurately at the RTL level. It helps to identify potential design defects.

Verilog HDL standard defines four states (1, 0, X, Z) to model various possible values of a signal. 'X' represents an unknown or indeterminate values in logic design. Indeterminate value could be introduced by different reasons, such as un-initialized signals, un-initialized memory, non-resettable flops, multiple drivers of a shared signal, timing violations on the flop, etc. X-prop plays a crucial role to propagate the ambiguous values in RTL and eventually those propagated values will be detected by the testbench. The 'X' might be caused by un-initialized value from the logic but RTL code and standard Verilog semantics often ignore the 'X', and conceal the logic bug which fails to handle the unintentional or indeterminate values.

This paper will discuss the advanced usage of X-prop on a real complex SoC design such as NXP LS1080. It will demonstrate X-prop configuration setup for Synopsys VCS, non-synthesizable module and non-resettable flop extraction with Synopsys VC-Static and defect analysis with Synopsys Delta Cycle and Verdi X trace.

2. X-Prop Model Review

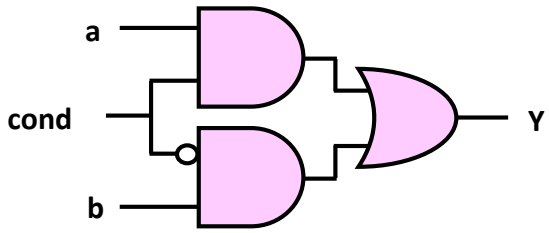
Synopsys VCS X-propagation simulator, or X-prop, provides more accurate semantics for RTL simulation. The X-prop model provides semantics to consider the effect of both 0 and 1 for every X-controlled assignment. It also employs a configuration mechanism to control pessimism/optimism.

VCS simulator provides the three merge modes of simulating RTL code, V-merge, T-merge and X-merge. T-merge and X-merge are particularly interesting in X-prop model:

1. V-merge mode: Normal RTL simulation
2. T-merge mode: Behavior closer to hardware, yields 'X' when output results would be different if the X were replaced with 0 or 1; this is the commonly used mode
3. X-merge mode: Yields 'X' whenever an input is 'X', which is as pessimistic as gate level simulation (or perhaps moreso)

The following example demonstrates the merge function for T-merge and X-merge in a simple if-else Verilog which models a 2-to-1 multiplexor. Figure 1 summarizes the merge function of T-merge and X-merge for the simple multiplexor, if input 'cond' is 'X'.

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



cond	a	b	Yrtl	T-merge	Ygate	X-merge
X	0	0	0	0	0	X
X	0	1	1	X	X	X
X	1	0	0	X	X	X
X	1	1	1	1	X	X

Figure 1. Truth table for if-else contruct with T-merge and X-merge

X-merge always generates 'X' on the output and is the most conservative or pessimistic mode. T-merge merges the outcome of the conditional being both 0 and 1, and gives a more releastic result than standard Verilog RTL in cases like this. The outcome is the merged result of executing all branches. In this case, Y will be the value of the merge of the values of 'a' and 'b'. If 'a' and 'b' have same value, Y will be that value. Otherwise Y will be 'X'. Note that the X-prop result may not be exactly the same as the gate result, since the gate result depends on the particular implementation of the mux function. The goal of X-prop is not necessarily to replicate the gate behavior, but it does help find X-related bugs in the design that would otherwise be masked by RTL X-optimism and only uncovered during gate simulation.

3. X-prop Configuration and Deployment

3.1 X-prop configuration

The X-prop simulation model could be compiled with or without an X-prop configuration file

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

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

The xprop.cfg configuration file specifies the modules and instances to which instrumentation is applied. Here, the configuration file is used to apply X-prop only to design code; no testbench code is included. In addition, non-synthesizable modules are excluded from X-prop instrumentation.

Here is the example code of xprop.cfg for NXP LS1080.

```
//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_cmplx } { xpropOff } ;
instance { testbench.top_ls1080.qb■an_c■plx } { 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 } ;
```

The testbench code is excluded in the X-prop model by turning off X-prop from the top including the testbench, then turning on at the DUT level (top_ls1080). X-prop could be turned off on any specific instance by specifying the full hierarchy of the instance and any module by module name.

3.2 VC-static to extract non-synthesizable modules

Detail examples are given in reference 4 on X-prop instrumentation of non-synthesizable code. In summary, Xs are expected to be generated from Latches, Behavioral memory models and testbench codes which bound to internal DUT modules[4]. X-prop will raise false alarm and causes false simulation failures due to the Xs propagating in non-synthesizable modules. VCS X-prop instrumentation does automatically exclude non-synthesizable code today, but our flow has adopted a standard practice of explicitly turning off X-prop on non-synthesizable modules.

VC Static is part of the Static and Formal tool suite from Synopsys. In this case, VC Static is used to find the non-synthesizable modules from the design. Most static/formal tools, including VC Static, compile the code and perform a synthesis-type transformation to work with the design. Since we are using VCS for simulation, compiling the same design in VC Static is straightforward, as both tools use the same compiler front-end. To compile in VC Static, we reuse the VCS compile options within the "read_file" command:

```
read_file -format sverilog -top <top> -vcs { <VCS compile options>}
```

After compiling the design, the "report_link" command gives the status of blocks that were compiled. The status codes "Synthesis," "UserBB," and "AutoBB" all indicate a non-synthesizable condition, so the blocks with these statuses are added to the xprop.cfg file to remove them from the X-prop instrumentation.

3.3 Xs on reset

The NXP SoC derives a pipelined version of an asynchronous reset from an active low power reset chip-level pin. The flops with this pipelined asynchronous reset are in an unknown state in the first a few cycles of simulation. At time 0, many signals are 'X' including the reset signal. The following example codes from the design shows there is an assertion to check the reset values in the default branch of one of these case statements. When X-prop is turned on, this always block is triggered at time 0 and multiple branches of case statements will be merged, which causes the assertion to fire. But at time 0 and the first few cycles before reset, this is more like a false alarm in this case.

```
always @(R_cur_pmgmt_cmd_invalid or
            R_cur_cdan_chid or
         ...
            R_rd_ewqdm or ...
            R_cur_wq_typ) begin : EWQM_PROCESS_CMD_asynch)
if (....)
.....
else if (R_rd_ewqdm[82:78] < EWQD_WQD_NUM_RA)
               begin
                  wr_ewqdm[82:78] = R_rd_ewqdm[82:78] + 'h1;
                  case (R_rd_ewqdm[82:78])
                     0: begin
                        if (0 >=EWQD_WQD_NUM_RA)
                        begin
                           wr_ewqdm[539:516] = 'h0; //COV_ILLEGAL IF ( 0 >= EWQD_WQD_NUM_RA)
                        end
                        else
                        begin
                           wr_ewqdm[539:516] = R_cur_enq_wqentry;
                        end
                     end
...
                     18: begin
                        if (18 >=EWQD_WQD_NUM_RA)
                        begin
                           wr_ewqdm[107:84] = 'h0; //COV_ILLEGAL IF ( 18 >= EWQD_WQD_NUM_RA)
                        end
                        else
                        begin
                           wr_ewqdm[107:84] = R_cur_enq_wqentry;
                        end
                     end
                     default:
                     begin
                        //  sv_pragma A_append_invalid_condition_2: assert (!reset_b) else ..;
                     end
                  endcase // caseR_rd_ewqdm[82:78]
               end
```

VCS provides the task $set_x_prop for user to turn on/off the X-prop sementics dynamically. In this case X-prop is turned off during this short period from time 0 until reset. Tasks $set_x_prop("vmerge") and $set_x_prop("tmerge") are used to setup regular RTL simulation and X-prop RTL simulation, respectively. The following codes shows the X-prop is turned off in the first 32 cycles until reset per design spec.

```
//This added for xprop
`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
  end
`else   //this is for IES
initial
  begin
   repeat (32) @ (posedge `SOC_DUT_TOP.SYSCLK);
   time0_done = 1;
  end
`endif

`endif //`ifndef GATE_MODEL
```

3.4 X-prop debugging

A primary reason for using the VCS X-prop simulation mode is to catch X's during RTL simulations, where debug is easier, instead of finding them only during gate simulations. But even in RTL, tracing X's presents a debug challenge. We found that debug features such as Verdi's Temporal Flow View and delta cycle expansion in both DVE and Verdi helped in tracing the cause of these X's.

Both DVE and Verdi provide an "Active X" trace feature which attempts to find the driver input which is responsible for the X output. Both tools account for the X-prop semantics when tracing active drivers, but Verdi provides an additional level of debug help for X-prop in its Temporal Flow View (TFV). The Verdi TFV provides an automated way to trace an X value back to its source, as shown in the figure below. This tool is helpful for debugging any X in simulation, but in an X-prop simulation, TFV's cycle-based schematic view adds a "t-merge" or "x-merge" annotation to indicate any part of the logic that is changed from standard RTL semantics. Verdi also adds a color code to the left side of the Source window, to indicate RTL code that is affected by the X-prop semantics.

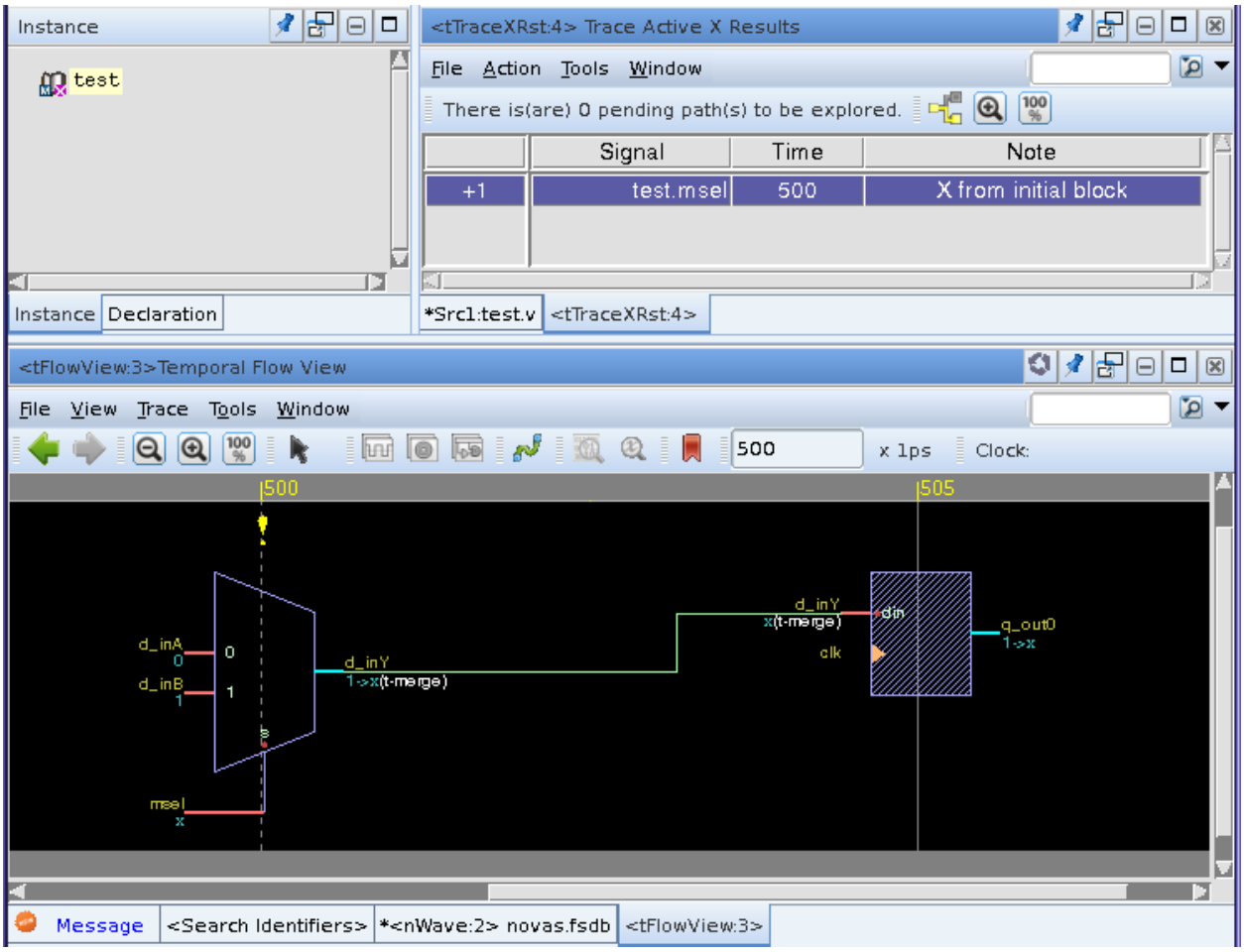


Figure 2. Verdi Temporal Flow View with X-prop

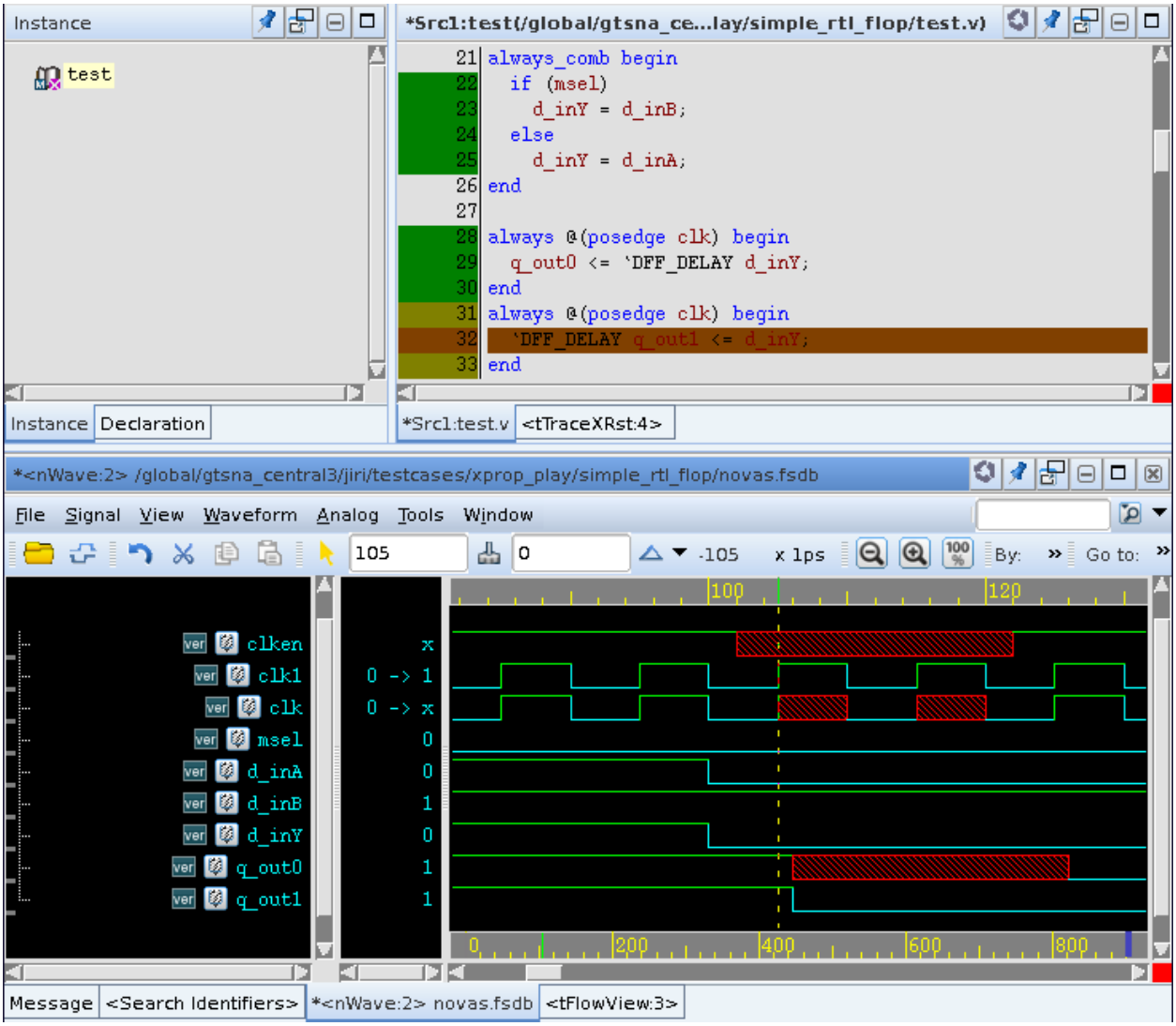


Figure 3. Verdi Source view with X-prop

In some cases, the X's that are generated are glitches that occur in zero time within a timestep of the simulation. With standard waveform dumping, when such X glitches occur, the X can show up further downstream, but its drivers appear to have no X's. To find the cause of the X in these cases, we enabled the delta cycle feature of the gui, which expands the timestep to show the order of all the delta events. The figure below shows an example of the delta cycle view in DVE, and a similar capability is available in Verdi. All of the edges within the shaded section of the waves are happening at the "same" simulation time, i.e. within a single timestep.

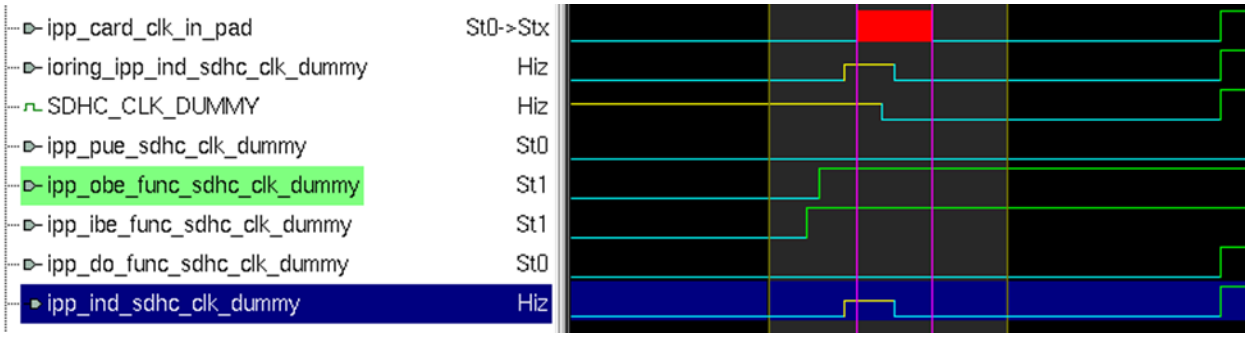


Figure 4. Delta cycle display in DVE

To view the delta cycle events in the gui, first the simulation must dump these delta events to the wave file. For the VPD waves used in DVE, the delta events are enabled by either using the verilog system task $vcdplusdeltacycleon, or with the UCLI command "dump -deltaCycle on". In the DVE gui, the delta events at the cursor time are expanded via the right-click menu selection "Expand Time". For the FSDB waves used in Verdi, the delta events are enabled with the simulation command-line options "+fsdb+glitch=0 +fsdb+sequential". Then, in Verdi nWave, the delta events at the cursor are shown with the "View | Expand Delta" menu selection.

4. Results

NXP LS1080 is complex networking SoC with 8 arm a53 cores, many high speed and low speed IOs, data path acceleration units, interconnect and NOCs and multiple SoC IP blocks including clocking, reset, power management and debug. Due to the complexity of the design and verification environment, X-prop has been deployed in a gradual manner starting from qualify regression to full regressions on different models.

A few design issues have been uncovered when the regressions were run on X-prop enabled design. Those issues were found during RTL simulation, but without X-prop could be only found on gate level simulation with correct test cases.

4.1 Un-driven pin and race condition (eSDHC case)

This test fails due to the X in data fifo. The X is traced back to ternary assignment and iomux logic, iomux_sdhc_dummy_clk, which is connected to pad in this example. The glitch on ipp_card_clk_in_pad is observed and it is caused by the race of simultaneous switching of signals.

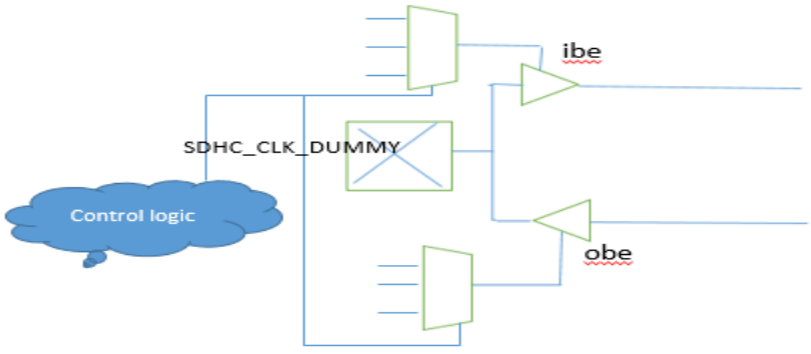


Figure 5. Simplified pad block diagram

SDHC_CLK_DUMMY has its IBE/OBE tied to 1'b1 in the IOMux logic, the tie constant values do not propagate until RCW is loaded to select eSDHC interface and when this occurs there is possible race condition on IBE/OBE which could cause a glitch on SDHC_CLK_DUMMY if IBE wins the race over OBE.

The following diagram with delta cycle expansion shows that the 'X' is because ibe is asserted before obe and the 'Z' on SDHC_CLK_DUMMY is captured in iomux logic then it is translate as glitch downstream logic.

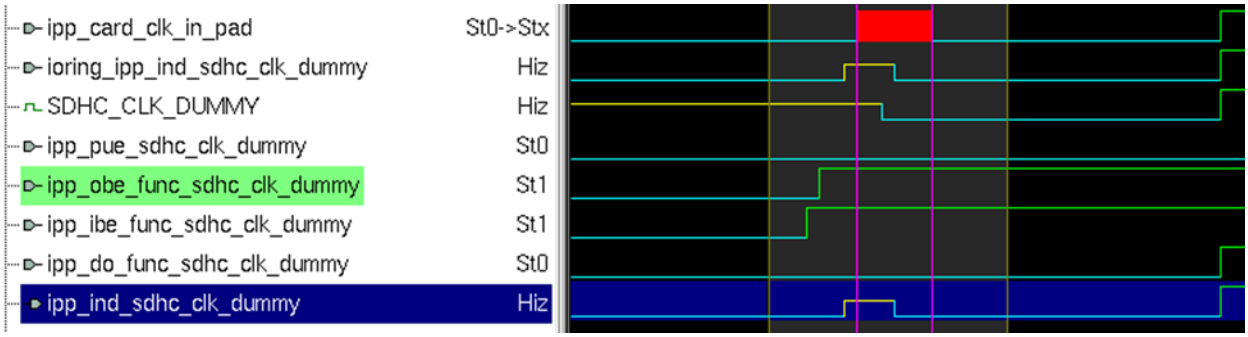


Figure 6. Delta-cycle view of IOMux signals

'X' on signal ipp_card_clk_in_pad is pipelined and propagated to clk signal ipg_clk_xmit of eSDHC asynchronous data fifo which eventually causes the esdhc fifo pointer to go to x.

```
// Transmit binary pointer register
ipd_ff_hold_b_macro
#(
   .WIDTH                          ( AWIDTH+1                          )
)
xmit_bin_ptr_reg
(
   .ipg_clk                        ( ipg_clk_xmit                      ),
   .ipg_hard_async_reset_b         ( ipg_hard_async_reset_xmit_b       ),
   .ipg_soft_reset_b               ( ipg_soft_reset_xmit_b             ),
   .hold                           ( xmit_bin_ptr_hold                 ),
   .data_in                        ( xmit_bin_ptr_ns[AWIDTH:0]         ),
   .data_out                       ( xmit_bin_ptr_ff[AWIDTH:0]         )
);
```

This test failure uncovered 2 design issues:

- SDHC_CLK_DUMMY pin is not driven.
- Race condition between ibe and obe in iomux logic.

The verification testbench adds a weak pull down on io_sdhc_clk_dummy pad as a workaround to address this issue on RTL simulation. The race condition was addressed at gate netlist by addressing timing of ibe/obe correctly.

4.2 Race condition on MUX -- WRIOP case

This test is hanging in X-prop simulation. When platform clk switches from sys_clk to platform clk, the X is generated from the following mux.

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

The output clock rgmii_refclk125t is used as trigger event for an always block,

```
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;
       if (count==6'h01)
          even <= 1'b0;
       if (count==6'h04)
       begin
          even <= 1'b1;
          {oldsel125,oldsel25} <= {sel125,sel25};
          clk2half <= 1'b1;
       end
    end
    if (~oldsel125&~oldsel25)
    begin
       count <= (count==6'd49) ? 6'h00 : count + 6'h01;
       if (count==6'd24)
         clk2half <= 1'b0;
       if (count==6'd49)
       begin
         clk2half <= 1'b1;
         {oldsel125,oldsel25} <= {sel125,sel25};
         even <= 1'b1;
       end
```

The following diagram with delta cycle expansion shows more details of this scenario. In this case, the 'X' on output rgmii_refclk125t is due to the simultaneously toggling of the 'sel' and 'data0' inputs of the clock multiplexor. Since rgmii_refclk125t is used as a trigger event for an always block, this 'X' triggered the always block as normal posedge in normal RTL simulation. However, in the X-prop simulation, the effects of the X representing both 0 and 1 are considered. For the always block, this means that the LHS results are the merged results of triggering and not triggering the always block.

In these two cases, the count value will either increment or not, so the merged value is X (on at least some of the bits). The even and clk25 signals also get X, which causes the test to hang.

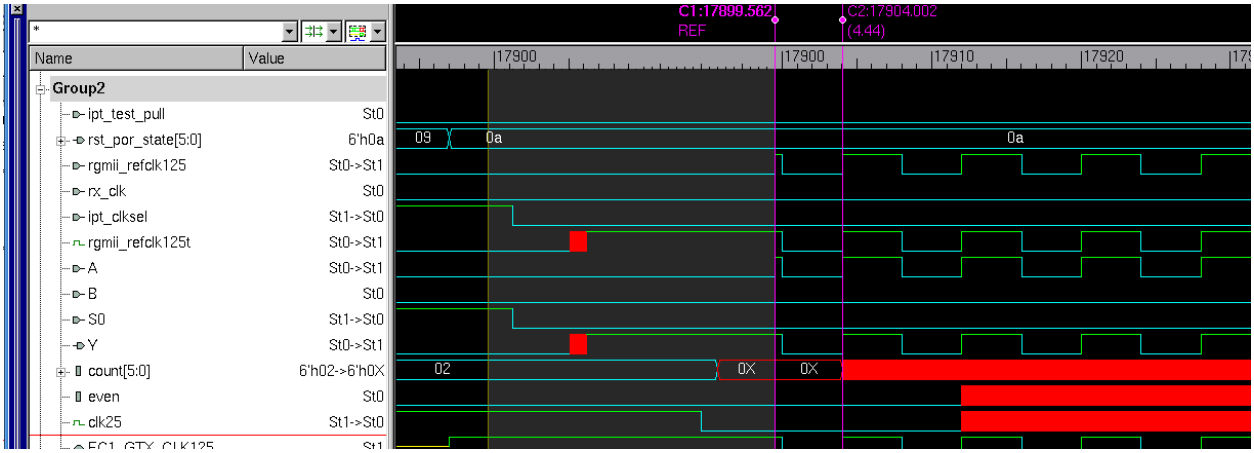


Figure 7. Delta-cycle view of rgmii_refclk125t and related signals

The root cause of this X was not found during the limited time available due to project schedules, but the effects of the X were thoroughly analyzed with the design team. In this case, the count value affected the pulse width of a divided clock during the initial startup of that clock. The resulting potentially shorter clock width at this phase of operation was determined to be harmless. The current DVE and Verdi debug tools, even with delta cycle expansion, could not provide enough trace information to root cause the source of the 'X'. Further debug of this scenario with Synopsys will continue, and a possible future enhancement of the tool may be required. The current workaround is turning off X-prop on this module.

The same race condition is also observed at gate level simulation, in which the glitch on the EC*GTX_CLK125 goes through the dcc logic and eventually to the counter which will latch the X. This scenario shows us that X-prop uncover the logic issue which could otherwise only be found by gate simulations, which are highly resource hungry and time consuming.

4.3 qDMA scenario

This qDMA failed test on X-prop provides the scenario that the downstream logic should be robust enough to handle the indeterminate value on its inputs.

In this specific scenario, the 'X' is generated from the following codes in NOC which is connected to qDMA directly.

```
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;
```

Signal Gnt doesn't get 'X' in regular RTL simulation but it has 'X' with X-prop because signal ReqArbIn (at NoC level) is 'X' in this case as this signal had indeterminate value at that point. This 'X' propagates to qDMA signal axi_bresp which cause the corruption of data inside qDMA. Since the below

assignment was continuous, any value of axi_bresp is reflected on signal data_rsp_fifo_bdone_wte_set.

Original equation:

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

From the equation, qDMA directly uses the BRESP signal to enable writing to the register holding the system bus error signals. BRESP could be indeterminate value when there is no valid. When valid is asserted, BRESP has determinate value. So valid should be used to qualify BRESP.

qDMA logic only puts response value to register when corresponding valid signal is asserted. Here is the updated equation to address the above issue.

New equation:

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

This protects the error register from being set continuously with unexpected data. Obviously the updated equation should fix the failure seen in X-prop mode.

Normal RTL simulation will skip this bug because axi_bresp has determinate values which will not fail the test. X-prop provides the mechanism where inputs could be driven as 'X' which eventually reveal the design defect.

5. Future Work and Enhancement

Non-resettable flip-flops will have indeterminate value at time 0. Those indeterminate values could be propagated as 'X' and raise the false alarm by failing the test. If the non-resettable flip-flops in the design could be extracted and be initialized with random value, it helps to remove the false alarm on the known 'X' origination. VC Static provides the mechanism to extract non-resettable flip-flops but the flow was not sufficiently developed during this project.

Debugging X-prop fails and analyzing the root cause of 'X' is not straightforward. In general, we compare the waveforms with X-prop and without X-prop of the same test. The delta-cycle feature in DVE/Verdi helps to see the hiding 'X' or glitch at problematic time stamp but it could not provide enough information for analyzing the root cause in the mux race case (Section 4.2). Any automation, quick trace mechanism and tool intelligence will be useful to allocate the bug quickly in this area.

6. Conclusions

This paper describes the methodology to use VC Static to extract non-synthesizable modules and apply them to xprop configuration and deployment. This practice saves a lot of debugging time and effort on false alarms and fails with X. Future development is expected to extend VC Static usage to help with non-resettable flops, and improve debug time still further. VCS X-prop mode helps to uncover design issues during RTL simulation which would otherwise only be found by gate level simulation.

7. References

[1] IEEE Standard for System Verilog. IEEE 1800-2012.

[2] X-Optimism Elimination during RTL Verification - SNUG Austin, 2012, Robert Booth et al, https://www.synopsys.com/news/pubs/snug/2012/austin/fa3_paper_booth.pdf

[3] Guide to the VCS X-Propagation Simulator, Synopsys

[4] A Practical Approach to Implementing Synopsys X-Prop Technology on a Complex SoC – SNUG Austin, 2013, Sounder Kumar, et. al, http://www.synopsys.com/news/pubs/snug/2013/austin/fa3_kumar_paper.pdf

When you are finished with your paper:

- ✓ Review and accept/reject all edits.
- ✓ Add or verify your paper title in the footer.
- ✓ Delete the Guidance section.
- ✓ Press Ctrl+A to select all of the text in your document and then press F9 to update all links. Select **Update entire table** on any prompt that appears.
- ✓ Turn on Track Changes (on the Review ribbon).
- ✓ Save your paper.