2020 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Build Reliable and Efficient Reset Networks with a Comprehensive Reset Domain Crossing Verification Solution

Wanggen Shi, Big Fish Semiconductor Ltd., China (*Shiwanggen@fishsemi.com*)

Yuxin You, Mentor, a Siemens Business, China (*yuxin_you@mentor.com*)

Kurt Takara, Mentor, a Siemens Business, USA (*kurt_takara@mentor.com*)

***Abstract*— Reset-related challenges and risks in design and implementation can compromise the intent of reliable and efficient reset networks in modem SoCs as the size and complexity of these SoCs continue to increase. These risks may not be immediately apparent and can result in unexpected chip behavior. Sequential elements reset by different asynchronous resets cause reset domain crossing (RDC) paths. Incorrect designs through these RDC paths will cause metastability and prevent designs from resetting to a known good state, which then results in unreliable behavior. This paper summarizes these challenges and risks, as well as introduces a methodology to build a structural reset network to address these challenges and risks. Finally this paper presents an effective and comprehensive verification solution, based on Mentor Questa RDC and used on real projects, to do reset checking statically, exhaustively and quickly, resulting in chips with efficient and reliable reset networks.**

Keywords*— *reset domain crossing, RDC, metastability, verification, reset ordering, reset synchronization

I. Introduction

Modern System-on-a-Chip (SoC) designs incorporate hundreds of Semiconductor Intellectual Property (SIP) blocks. This high use of SIP, each with their own reset requirements, combined with (among other things) needs for deterministic behavior at reset related to system reliability, result in complex reset requirements. The complex reset architectures that emerge to satisfy these many requirements result in many asynchronous reset domains, and functional reset ordering requirements and assumptions [1]. Data and control signals cross from one sequential element to another between these asynchronous reset domains thereby causing reset domain crossing (RDC) paths [1][2]. For RDC paths, an asynchronous reset at a transmitting register may cause a receiving register to sample an asynchronous event and become metastable. This metastability will cause unpredictable values to be propagated to down-stream logic and prevent a design from resetting to a known good state which results in the device working unreliably. In the worst case, reset issues may occur on many RDC paths and cause a device to consume excessive power at assertion of reset, causing the device to be permanently damaged.

In many of today's design projects, there is a gap between the design and verification of reset architectures. The objective of the reset architecture is to initialize every sequential element in the design directly or indirectly, so the RDC design requirements specification is a critical part of the reset architecture design process. However, ensuring that every RDC is verified for proper data and/or control signal behavior requires more than just a good design review. If the goal is silicon that works properly and deterministically, care must be taken to verify the reset architectures and the RDCs thoroughly and completely.

II. What is the RDC Problem?

Similar to Clock Domain Crossings (CDCs), Reset Domain Crossings divide the designs into reset domains. Each reset domain is reset independently from other reset domains [2]. Transmission of data or control signals between two reset domains is termed a Reset Domain Crossing.

Figure 1 illustrates a simple RDC problem. Reset signal rst1 (active low) is asserted asynchronously. This causes the output 'q1' of register 'R1' to change. This change to 'q1' could have a material effect on the 'd2' input of register 'R2'. If the assertion of 'rst1' occurs at the right time such that the change to 'd2' occurs close enough to a rising clock edge on register 'R2' of the shared clock 'clk', the setup or hold time requirements for the register 'R2' can be violated. When this occurs, 'R2' can enter a transitional, indeterminate state, known as a metastable state.

This behavior is familiar to sequential logic designers who address asynchronous clock domains. There are well-known design techniques that are used to isolate the consumption of the transitional state during its probabilistic duration, so as to prevent corruption of data or control signals downstream. However, this same level of diligence is often not given to reset domain crossings. In particular, note that this scenario is not covered by traditional clock domain crossing analysis as the clock for both the transmit register 'R1' and the receive register 'R2' is shared. There is no CDC here as the path is synchronous. It is easily overlooked as a source of data or control corruption unless specific reset analysis is performed.

III. Design Solutions to the RDC Problem

Just as with Clock Domain Crossings, there are multiple methods that can be used to ensure that the downstream logic from an RDC does not attempt to consume corrupted data or control signals. These include the use of synchronizers and isolation strategies. However, in addition to the mechanisms available for proper CDC designs, RDC designs have an additional mechanism available, that of the reset sequence constraint.

A. *Synchronization Techniques*

The traditional means to resolve metastable signaling is accomplished through the use of a second (or more) serial register(s) on the receiving register in the RDC. By combining two consecutive registers, this significantly reduces the probability of a metastable signal reaching the consuming logic to something well beyond a traditional lifetime for most silicon technologies and designs and is generally considered acceptable. However, this also creates additional latency which may be considered undesirable.

B. *Isolation Techniques*

Isolation techniques rely on the creation of a valid, or isolation, signal that indicates to the register in the receiving domain when it is acceptable to sample the data. This signal therefore isolates the receiving registers. This can be done by gating the clock as shown in Figure 2. In this scenario the isolation signal 'isolation_sig' is asserted (active low) prior to the assertion of the 'rst1' asynchronous reset (active low) in the transmitting domain. This prevents the gated clock 'gated_clk' from rising and prevents the update of the register 'R2'. As noted in Figure 2, the 'R2' output 'q2' remains stable.

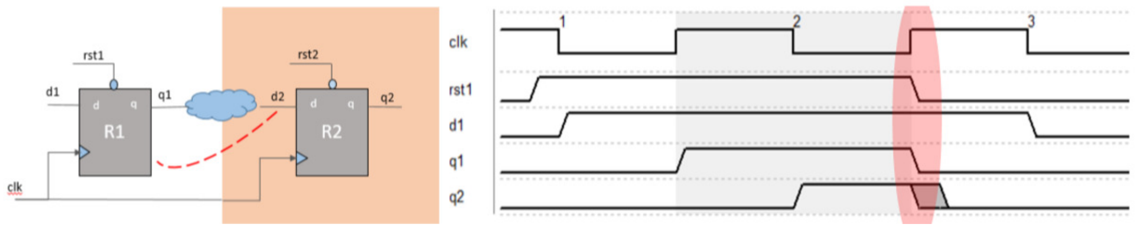


Figure 1. Potential Metastability at the Receiving Register of the RDC

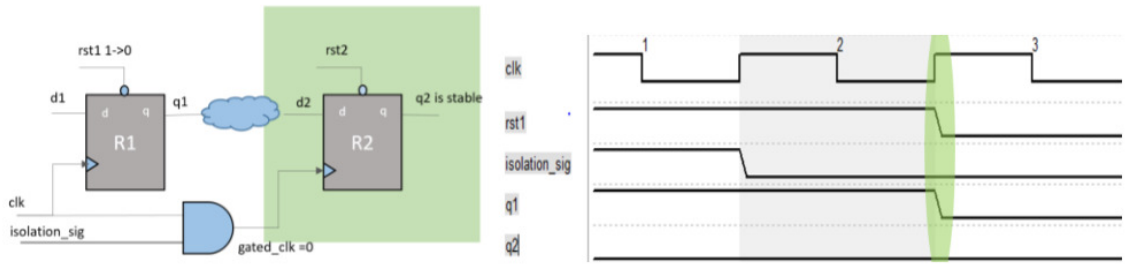


Figure 2 Example of clock gating-based isolation for a valid RDC

This same result can be accomplished through the use of a multiplexer on the 'd' input of the receiving register 'R2' that selects between the 'q' output of 'R2' or the information coming from the transmit domain to 'R2'. This multiplexer would hold the 'd' input as the value on the 'q' output when the 'isolation_sig' is active (low). In this manner, the receiving register 'R2' only updates when the transmitting side indicates that it is valid to do so with the de-assertion of 'isolation_sig'.

C. Reset Sequencing:

Synchronization or isolation techniques are common, effective means for both CDC and RDC design. However, RDC designs have another mechanism available by which an RDC can be valid, that of the reset sequence. Reset sequencing is either a consequence of the reset architecture, or an architectural choice. In the event that the receiving domain on the RDC is able to be put into reset, and held in reset, prior to the assertion of the reset in the transmit domain, there will be no metastable event. This is illustrated in Figure 3. Reset sequencing is able to render an RDC valid since the receiving register 'R2' is held in reset (via the earlier assertion of its reset on signal 'rst2') when the 'q1' output of the transmitting register 'R1' transitions due to the assertion of 'rst1'. Therefore in this scenario there will be no metastable behavior on register 'R2'.

IV. Verification Methodology For Valid Reset Domain Crossings

With a set of valid RDC design techniques available for use, SoC development teams should be able to reduce the occurrence of reset-related issues in silicon that result in unreliable use, or worst-case in damage to the component. However, it is always necessary to verify that the proper design techniques have indeed been used and that the constraints for the analysis are valid.

Functional simulation is insufficient for this task as it requires that just the right timing combination of an asynchronous reset and clock be present to create the metastable condition. It also requires that metastable behaviors are enabled in simulation. Finally it requires that the output of the register in a metastable state is sensitized through downstream logic such that it is detectable to the testbench pass/fail mechanisms. Running large numbers of simulation cycles hoping for just the right combination to occur is wasteful at best and impractical at worst. Exhaustive formal-based verification mechanisms such as those used by Questa® RDC from Mentor, a Siemens Company, address this issue.

RDC verification tools such as Questa RDC employ static and formal methods to analyze the design, detect asynchronous reset domain crossings, and verify that the proper structures are in place to ensure that the data or control signals will cross reliably. Should this not be clear to the tool, constraints can be applied, such as identification of isolation enable signals, or reset sequence constraints, which would allow the tool to determine that a crossing is valid, such as in Figure 2 or Figure 3.

However, a quality run of an exhaustive, formal-based RDC verification run relies on quality constraints and clean reset network definitions. In order to ensure the fastest, highest quality analysis and lowest noise possible in the formal RDC analysis, quality RDC tools such as Questa RDC enable static checks on reset networks prior to the formal verification steps. Performing these static checks before doing the full formal RDC verification help develop constraints and ensure that the reset networks are as clean as possible for formal analysis. For example, these static checks look for any potential transitional glitches in the design. While some of these issues may be detected in functional verification with timing delays, or in static timing analysis (STA) for a particular implementation, this does not enable the broad SIP re-use necessary in modern SoC developments. In addition, the use of static verification on designs to exhaustively detect glitch and other reset network quality/intent issues ensures a quality RDC verification run.

As for quality constraints, a key difference between synchronization and isolation techniques or reset sequencing is that the isolation enables or reset sequencing are defined by constraints, not derived by analysis. As a result, it is a best practice to verify the constraints. For isolation enables, this is done automatically by the tool with a functional check in the design to verify that the isolation enable can truly isolate. For reset sequencing, this is best accomplished by creating an assertion that can be proven formally or in simulation that will ensure that the constraint applied to remove the RDC from evaluation is indeed valid.

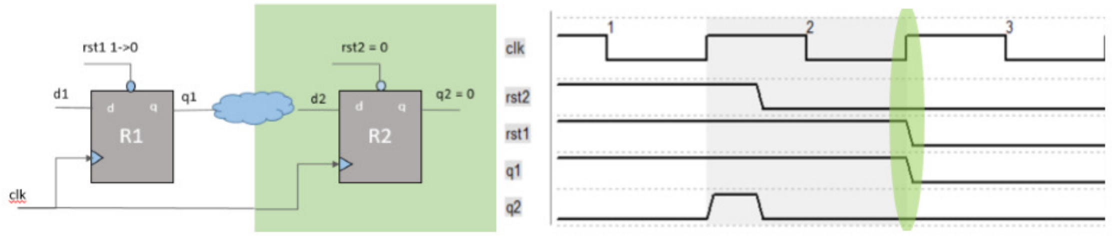


Figure 3 Example of reset sequencing for a valid RDC

These elements form an RDC verification methodology that can be easily used to ensure quality silicon. This methodology is specified in Figure 4. Static checks should be run on the RTL first until the RTL is clean. Following that, the formal RDC analysis should begin, bringing in constraints. If the results are not clean, either the RTL or the constraints should be revisited until the RDC runs are complete and clean. In parallel, the constraints should be independently verified.

V. EXAMPLES OF RESET NETWORK ISSUES IN DESIGN CAUGHT BY RDC VERIFICATION METHODOLOGY

A. *Glitches found by static checks*

An issue was found in our design by the static checks that verify the reset network in the Questa RDC tool. The issue was identified in code similar to that in Figure 5. Differing delays between bit 1 and bit 0 of the signal 'c_st' on the transition between 'IDLE' and 'WAKE' could potentially create a glitch on the signal 'work_flag_rstn'. This would occur if the 'c_st' value momentarily was equivalent to 'PD'. In that event, the signal 'work_flag_rstn' would assert briefly, inadvertently and unexpectedly resetting the 'work_flag' and "active_r' registers.

Reinforcing the value of these static checks prior to running the exhaustive formal analysis necessary for complete RDC verification, this glitch should be eliminated before it is potentially considered as a source of reset which will appear asynchronous to the clock 'clk'.

B. *Asynchronous FIFO RDC Issue*

We instantiated a commonly-available dual-port FIFO in our design. The design was an asynchronous FIFO but we intended to use it synchronously. Because of this, no asynchronous reset issues were raised in our CDC analysis. However, upon running the RDC verification (that we learned was critical for our success), Questa RDC identified an RDC issue through the FIFO because the two resets on each side of the FIFO did not have synchronous constraints and were therefore asynchronous, and because the reset ordering that was part of the reset architecture had not been specified. This pessimistic analysis was useful as it let us realize the need for a constraint to match the reset architecture, as well as the need for an assertion for formal or simulation-based verification to ensure the constraint was valid.

VI. RESULTS

The results of the static analysis and glitch detection are shown in Figure 6. This issue was found very late in our project development via functional simulation. While we were fortunate to find it at all given the timing-sensitive nature of the transition, the value of these complete and timing-insensitive checks early in the development process when most easily fixed, is apparent.

The results of the RDC run prior to adding the reset ordering constraint are shown in Figure 7. While in the end the reset architecture would have prevented the metastability in the FIFO, the thorough analysis, followed by review of the architecture, creation of constraint and then assertion-based verification of the constraint provided for highest confidence that nothing was missed.

We achieved our goal of reliable silicon operation.

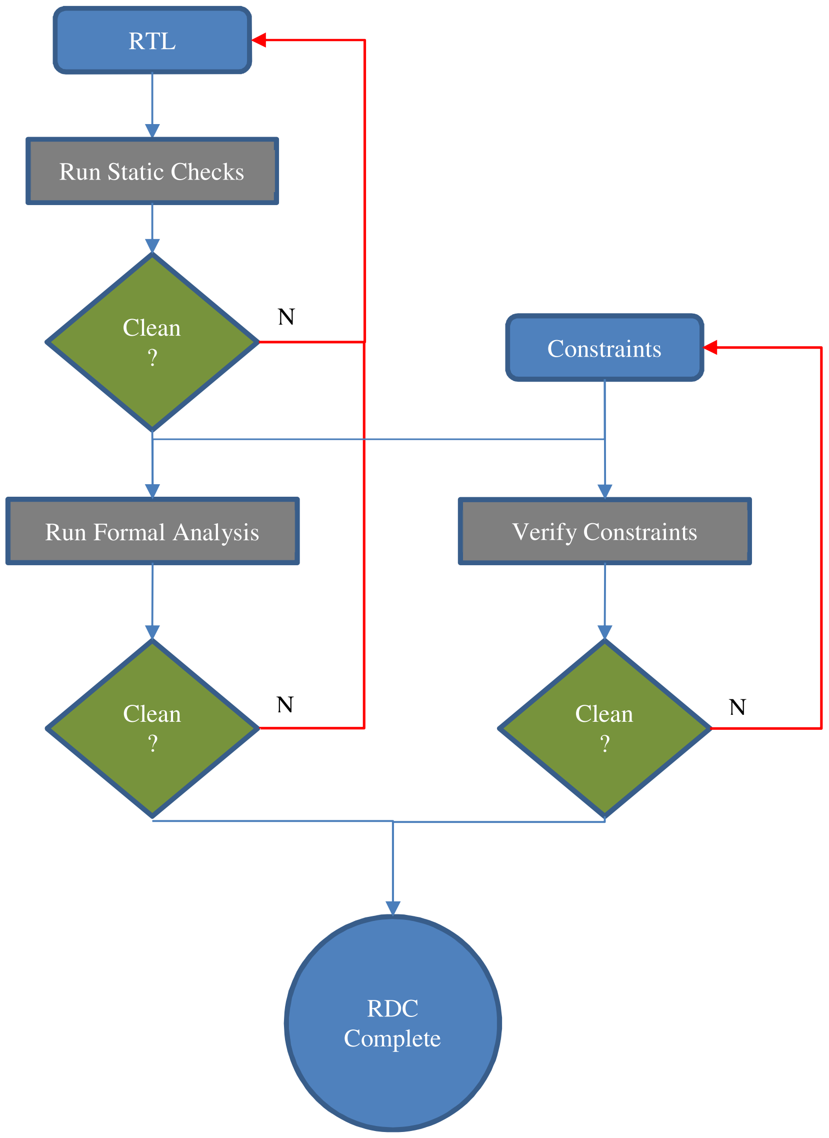
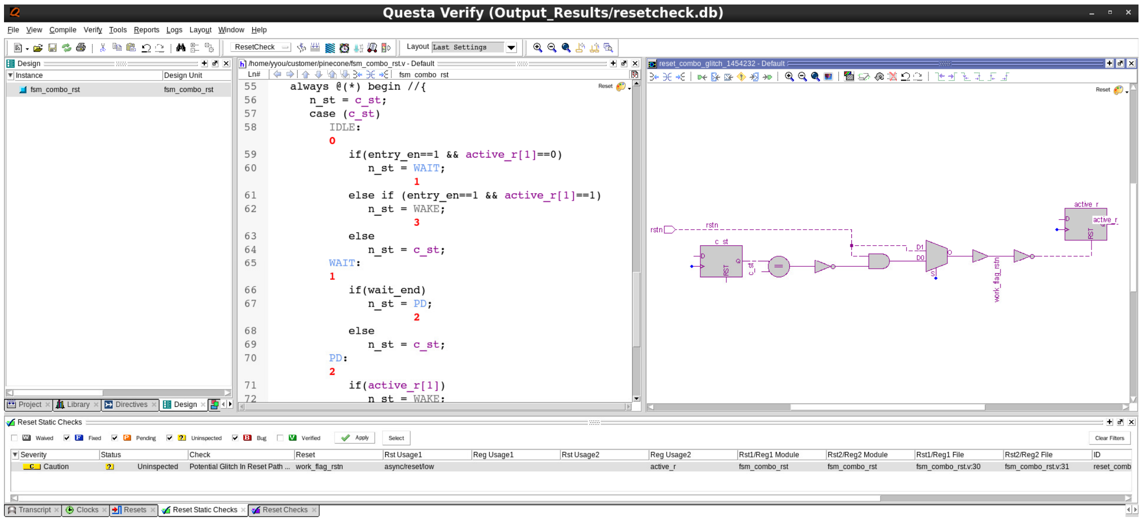


Figure 4 Recommended RDC Verification Methodology

```
module fsm_combo_rst (/*autoarg*/
   // Outputs
   work_flag,
   // Inputs
   clk, rstn, entry_en, active, wait_end, wake_end, work_en, test_mode
   );

   parameter   IDLE = 2'b00;
   parameter   WAIT = 2'b01;
   parameter   PD   = 2'b10;
   parameter   WAKE = 2'b11;

   input    clk         ;
   input    rstn        ; // clock_domain=clk
   input    entry_en    ; // clock_domain=clk
   input    active      ; // clock_domain=async
   input    wait_end    ; // clock_domain=clk
   input    wake_end    ; // clock_domain=clk
   input    work_en     ; // clock_domain=clk
   input    test_mode   ;

   output   work_flag;

   reg   [1:0] c_st;
   reg   [1:0] n_st;
   reg         work_flag;
   wire        work_flag_rstn;
   reg   [1:0] active_r;

   assign work_flag_rstn = test_mode ? rstn : rstn && !(c_st==PD);
   always @(posedge clk or negedge work_flag_rstn) begin//{
      if(!work_flag_rstn) begin//(
         work_flag <= 0;
         active_r  <= 0;
      end //)
      else begin//[
          active_r  <= {active_r[0],active};
          if (work_en)
             work_flag <= 1;
      end //]
   end //always @}

   always @(posedge clk or negedge rstn) begin//{
      if(!rstn)
         c_st <= IDLE;
      else
         c_st <= n_st;
   end //always @}

   always @(*) begin //{
      n_st = c_st;
      case (c_st)
         IDLE:
            if(entry_en==1 && active_r[1]==0)
               n_st = WAIT;
            else if (entry_en==1 && active_r[1]==1)
               n_st = WAKE;
            else
               n_st = c_st;
         WAIT:
            if(wait_end)
               n_st = PD;
            else
               n_st = c_st;
         PD:
            if(active_r[1])
               n_st = WAKE;
            else
               n_st = c_st;
         WAKE:
            if(wake_end)
               n_st = IDLE;
            else
               n_st = c_st;
      endcase
   end //always @}
endmodule
```

Figure 5 SystemVerilog code example illustrating a potential reset glitch


Questa Verify (Output_Results/resetcheck.db)

/home/yyou/customer/pinecone/fsm_combo_rst.v - Default

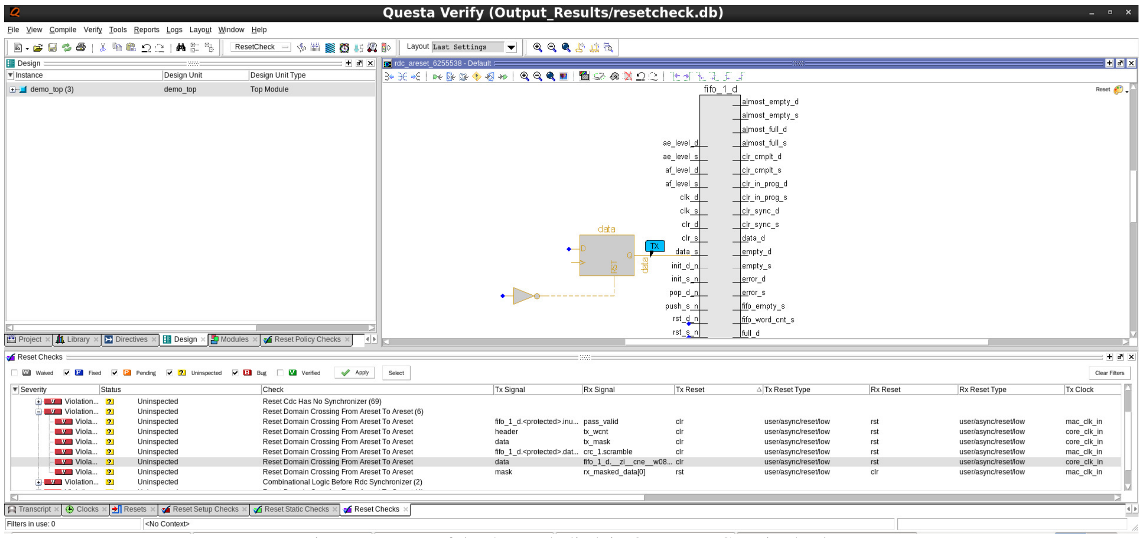
```
    always @(*) begin //{
       n_st = c_st;
       case (c_st)
          IDLE:
          0
             if(entry_en==1 && active_r[1]==0)
                n_st = WAIT;
                      1
             else if (entry_en==1 && active_r[1]==1)
                n_st = WAKE;
                      3
             else
                n_st = c_st;
          WAIT:
          1
             if(wait_end)
                n_st = PD;
                      2
             else
                n_st = c_st;
          PD:
          2
             if(active_r[1])
                n_st = WAKE;
```

Reset Static Checks

Severity	Status	Check	Reset	Rst Usage1	Reg Usage1	Rst Usage2	Reg Usage2	Rst1/Reg1 Module	Rst2/Reg2 Module	Rst1/Reg1 File	Rst2/Reg2 File	ID
Caution	Uninspected	Potential Glitch In Reset Path ...	work_flag_rstn	async/reset/low			active_r	fsm_combo_rst	fsm_combo_rst	fsm_combo_rst.v:30	fsm_combo_rst.v:31	reset_comb

File View Compile Verify Tools Reports Logs Layout Window Help
ResetCheck
Layout Last Settings
Design
Instance
Design Unit
fsm_combo_rst
reset_combo_glitch_1454232 - Default
rstn
c_st
RST
active_r
work_flag_rstn
Waived Fixed Pending Uninspected Bug Verified Apply Select Clear Filters
Project Library Directives Design
Transcript Clocks Resets Reset Static Checks Reset Checks

Figure 6 Report of the detected glitch in Questa RDC static checks


Questa Verify (Output_Results/resetcheck.db)

Reset Checks

Severity	Status	Check	Tx Signal	Rx Signal	Tx Reset	Tx Reset Type	Rx Reset	Rx Reset Type	Tx Clock
Violation...	Uninspected	Reset Cdc Has No Synchronizer (69)							
Violation...	Uninspected	Reset Domain Crossing From Areset To Areset (6)							
Viola...	Uninspected	Reset Domain Crossing From Areset To Areset	fifo_1_d.<protected>.inu...	pass_valid	clr	user/async/reset/low	rst	user/async/reset/low	mac_clk_in
Viola...	Uninspected	Reset Domain Crossing From Areset To Areset	header	tx_wcnt	clr	user/async/reset/low	rst	user/async/reset/low	core_clk_in
Viola...	Uninspected	Reset Domain Crossing From Areset To Areset	data	tx_mask	clr	user/async/reset/low	rst	user/async/reset/low	core_clk_in
Viola...	Uninspected	Reset Domain Crossing From Areset To Areset	fifo_1_d.<protected>.dat...	crc_1.scramble	clr	user/async/reset/low	rst	user/async/reset/low	mac_clk_in
Viola...	Uninspected	Reset Domain Crossing From Areset To Areset	data	fifo_1_d.__zi__cne__w08...	clr	user/async/reset/low	rst	user/async/reset/low	core_clk_in
Viola...	Uninspected	Reset Domain Crossing From Areset To Areset	mask	rx_masked_data[0]	rst	user/async/reset/low	clr	user/async/reset/low	mac_clk_in
Violation...	Uninspected	Combinational Logic Before Rdc Synchronizer (2)							

Filters in use: 0 <No Context>

Figure 7 Report of the detected glitch in Questa RDC static checks

VII. Conclusion

With the increase in asynchronous resets in today's hardware designs, RDC verification becomes a much more difficult task. A methodology is required to make sure RDC verification is complete and efficient. This methodology consists of more than just running the RDC tool on the design. It involves running static checks first on the design to ensure that the formal runs are of the highest quality (and yield). Our example of the glitch needing to be removed before formal analysis, which would interpret this as a reset, is a prime example. It also involves ensuring that constraints applied in reset ordering schemes are validated via assertion-based formal or simulation verification. Our example of the dual-port FIFO is an example.

In addition, the RDC methodology quantifies each step in the verification process and enables design teams to quantitatively demonstrate verification completion. Using this comprehensive verification solution allows teams build a reliable reset network efficiently.

VIII. References

[1] Chris Kwok, Priya Viswanathan, Ping Yeung, “Addressing the Challenges of Reset Verification in SoC Designs”, DVCon US, 2015.

[2] Yossi Mirsky, “Comprehensive and Automated Static Tool Based Strategies for the Detection and Resolution of Reset Domain Crossings”, DVCon US, 2016.