2015 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

# Addressing the Challenges of Reset Verification in SoC Designs

Chris Kwok

Priya Viswanathan

Ping Yeung

Mentor Graphics

# The Reset Problem

- There are many types of resets:
  - Power-on resets, software resets, hardware resets, debug reset, etc.
- Errors in reset can lead to metastability, glitches, corrupt simulation with X or even failure to power on

# Current Methods

- Simulation, simulation, simulation
- Issues usually found at late-stage, after long gate-level simulation

# Purpose

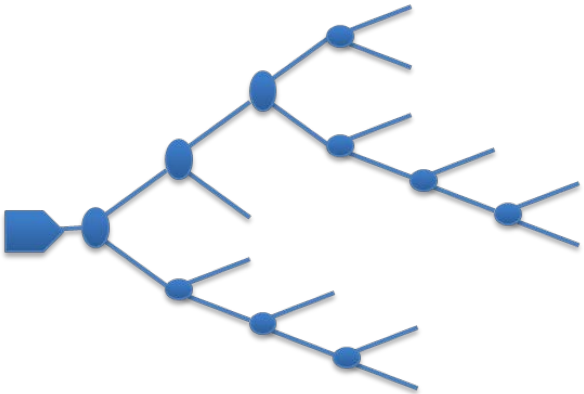
- Bring awareness of the reset problem
- What kind of reset problems exist
- What kind of solutions available

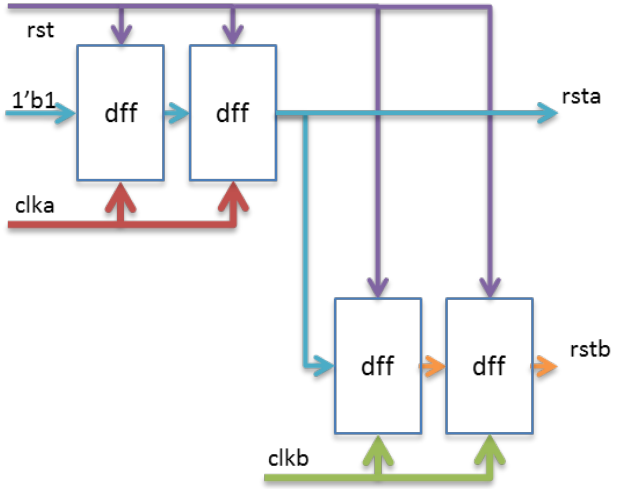
# Common Problems

- Reset Distribution Tree
  - Reset tree construction issues
- Reset Usage
  - Issues with how reset is used

# Reset Tree Problem

- Mixed asynchronous/synchronous types
- Overlapping set and reset
- Reset crossing clock domains





# Reset with mixed types

- Signal ‘rst_n’ is used as asynchronous to ‘q1’ and synchronous to ‘q2’
- Usually indicates a misunderstanding of the reset

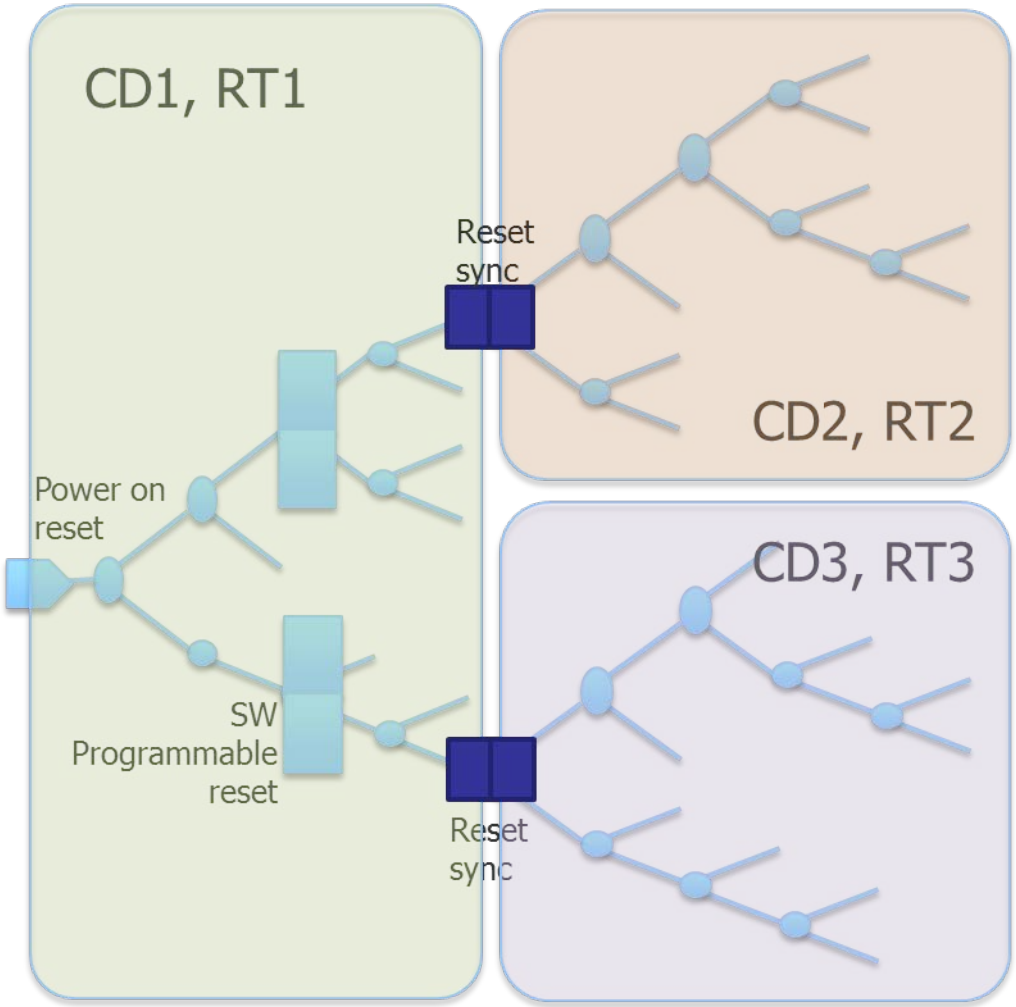
```
always @(posedge clk or negedge rst_n)
    if (!rst_n)
        q1 <= 1'b0;
    else
        q1 <= d;

 always @(posedge clk)
    if (!rst_n)
        q2 <= 1'b0;
    else
        q2 <= d;
```

# Overlapping set and reset

- Register has both asynchronous set and reset signals
- If both set & reset are active, lead to mismatch between simulation & synthesis
- Some technology libraries do not have register with both asynchronous set & reset

```
always @(posedge clk or negedge rst_n or negedge set_n)
    if (!rst_n)
        q1 <= 1'b0;
    else if (!set_n)
        q1 <= 1'b1;
    else
        q1 <= d;
```

# Reset crossing clock domains



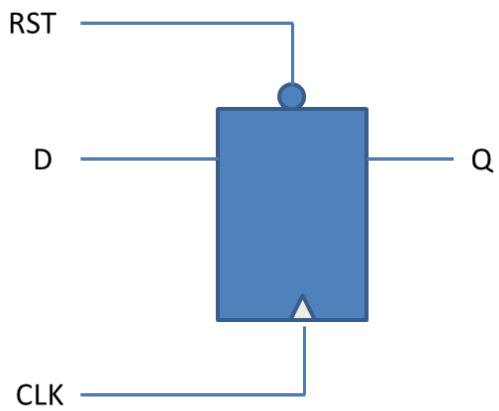

- The reset tree spans multiple clock domains
- Reset signals need to be synchronized to the target clock domain before used.

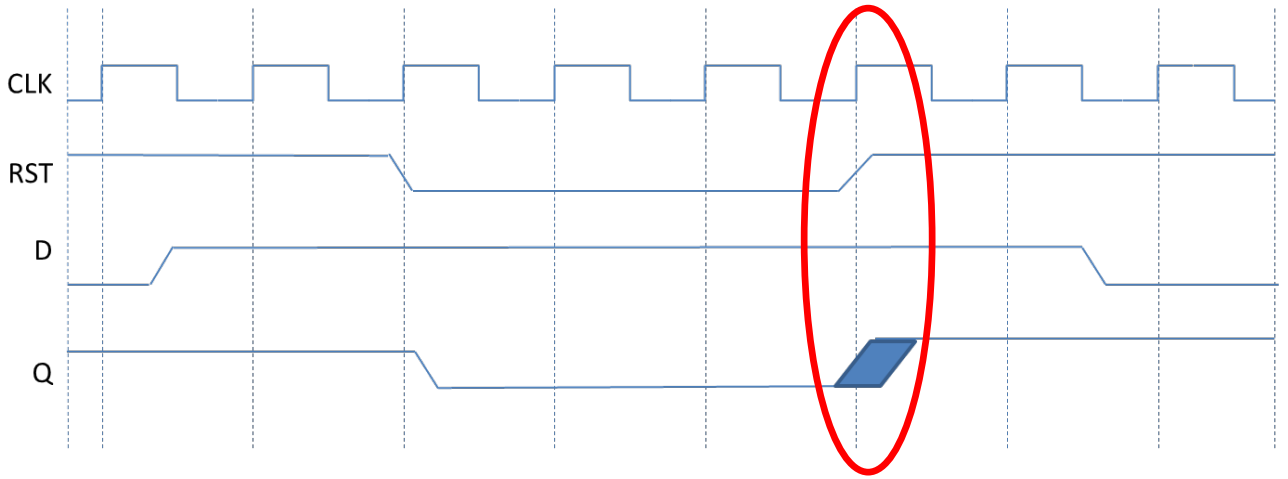
# Reset Synchronizer

- Why do we need reset synchronizer?
- How is reset synchronizer used?
- What other problems related to reset synchronizer?

# Asynchronous Reset

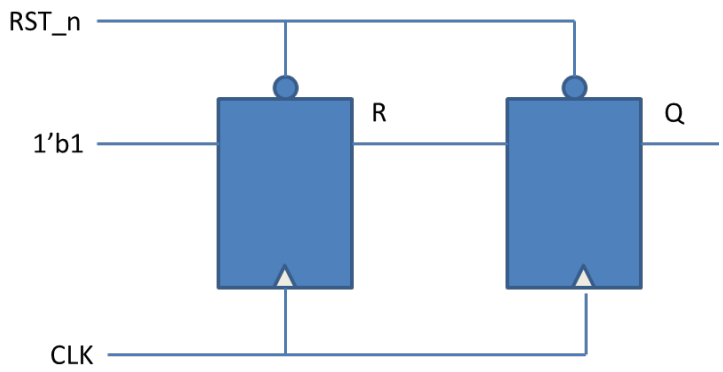
- Asynchronous reset signal can change anytime
- If reset ‘RST’ is de-asserted close to the clock edge
  - Violates the recovery time
  - Leads to metastability

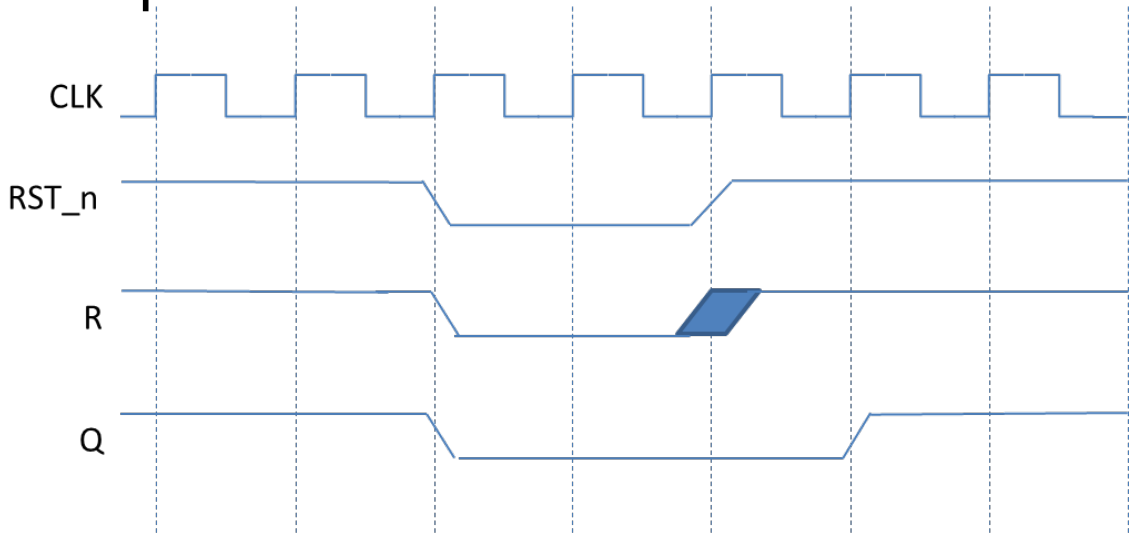


# How to fix it?

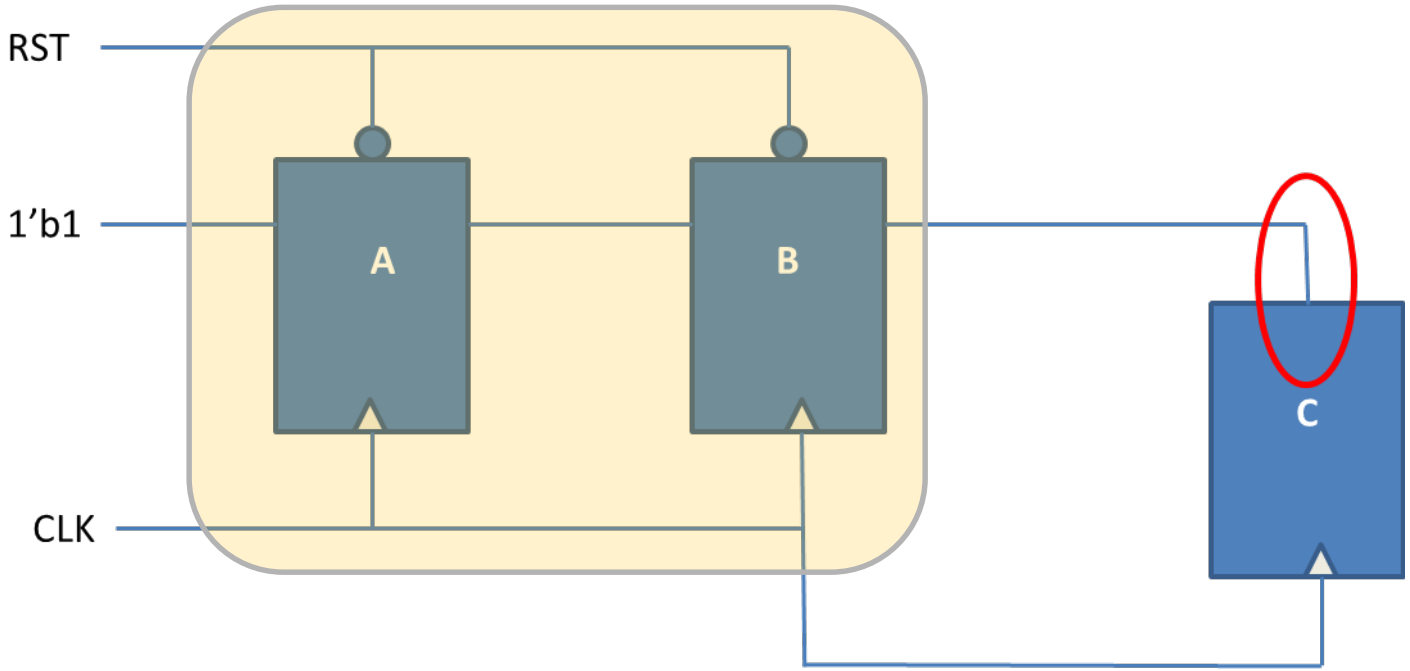
- Synchronize before use
  - Typical synchronizer is 2 back-to-back flops
- Ensure asynchronous assertion and synchronous de-assertion
- X will not be seen at the output of the synchronizer



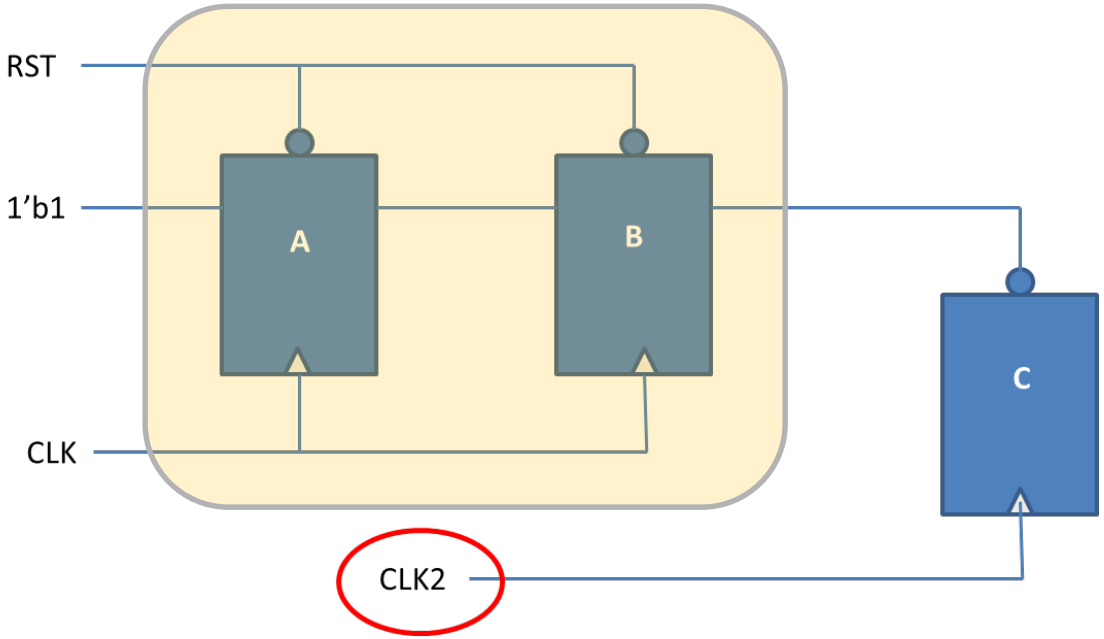
# Reset Synchronizer used wrong polarity

- Register reset by the synchronizer is using active high reset, while the synchronizer is an active-low reset.



# Reset Synchronizer used with wrong clock

- Downstream register using reset synchronizer is clocked in a different clock (CLK2) from reset synchronizer (CLK)
- Caught by CDC tool

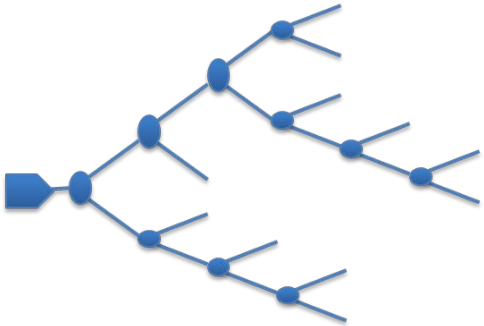


# Reset Domain Crossing

- What is reset domain?
- What is reset domain crossing (RDC) ?

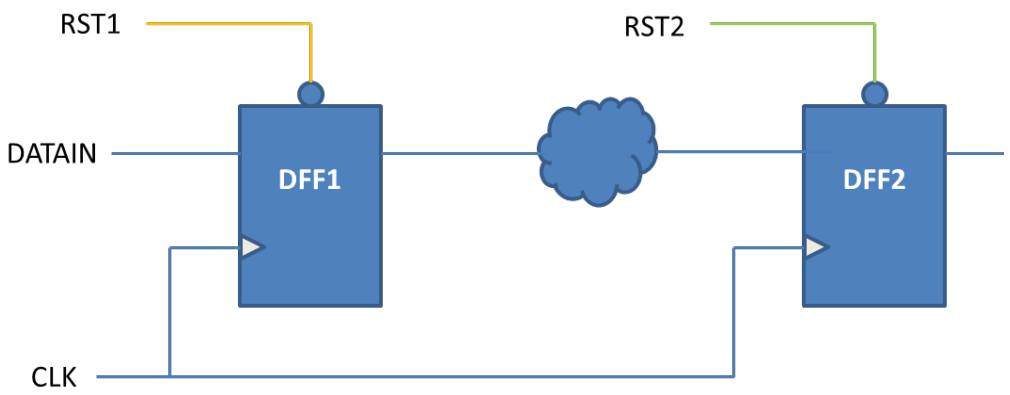
# What is a Reset Domain?

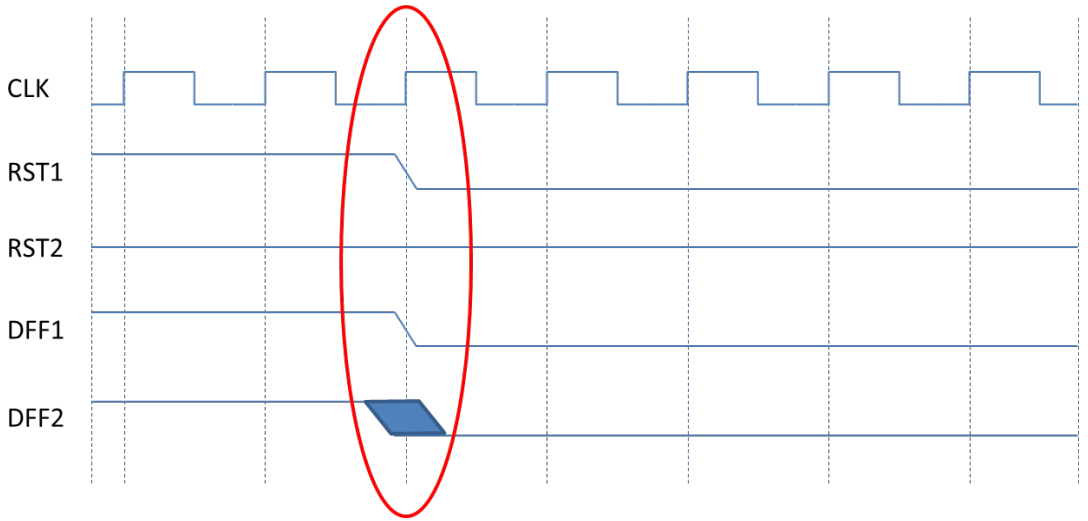
- What constitute a reset domain?
  - Type: Synchronous / Asynchronous
  - Polarity : Active-high / Active-low
  - Value : 1 (set) / 0 (reset)
- Registers reset by reset of the same attributes are considered to be in the same reset domain



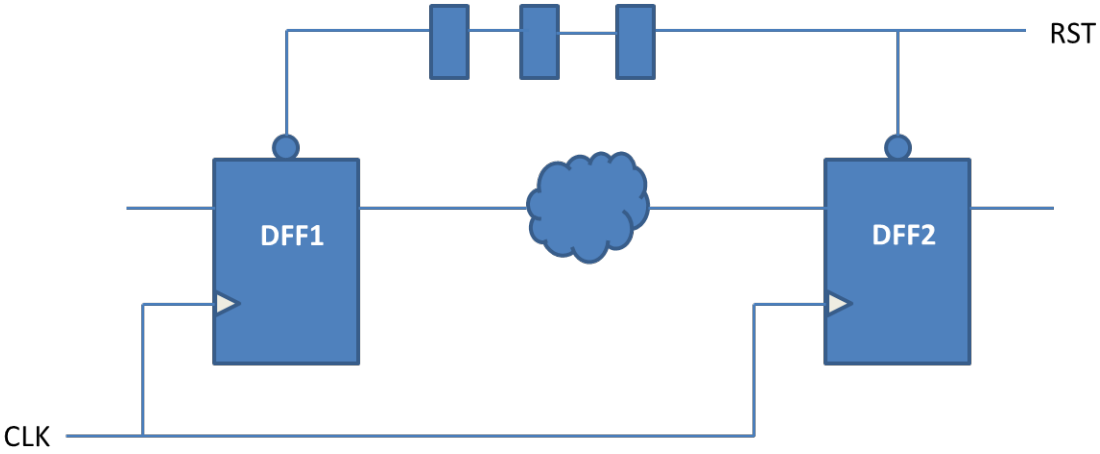
# What is Reset Domain Crossing (RDC)?

- If RST1 is asserted while RST2 is not asserted, DFF2 can sample corrupted data.



# RDC – Incorrect Delay Sequences




- Assertion of RST will prematurely cause DFF2 to reset while DFF2 is still in functional state.
  - Causes incorrect data at the downstream logic
- If RST is only asserted 2 cycles, then when RST deasserts, DFF2 will be corrupted by DFF1 before DFF1 gets the reset signal.

# Proposed Verification Solution

- Static Analysis
  - Automatically analyzes the design and report obvious errors
- Simulation with X-propagation
  - Propagate X values to cause testbench failure
- Formal Verification
  - Exhaustive Analysis on special properties

# Design Blocks

| Design Complexity | Block 1 | Block 2 | Block 3 |
|---|---|---|---|
| Number of registers | 305 | 47016 | 43622 |
| Number of latches | 0 | 592 | 0 |
| Number of RAMs | 2 | 0 | 64 |
| Number of Gate-level modules | 0 | 88 | 20 |

| Reset/Clock domains information | Block 1 | Block 2 | Block 3 |
|---|---|---|---|
| Number of reset domains | 3 | 36 | 48 |
| Number of clock domains | 9 | 5 | 12 |
| Number of clocks crossing reset domains | 2 | 3 | 6 |
| Number of resets crossing clock domains | 2 | 5 | 5 |

- Verilog RTL design blocks
- FIFO Controller, Functional Controller, Networking Unit

# Reset/Clock Relationship

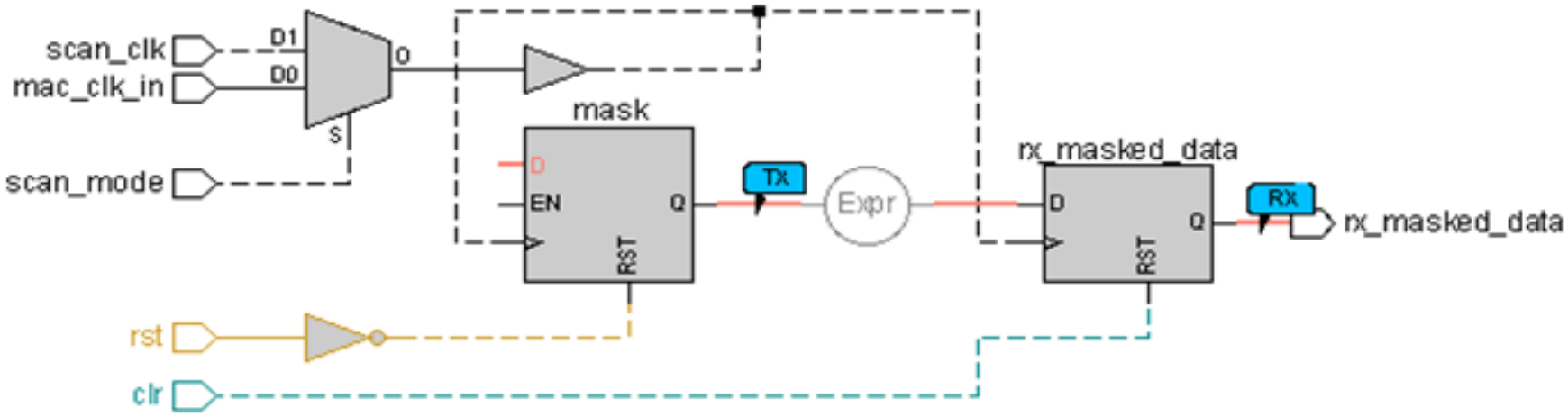
- Questa Reset Check was used

# Static Analysis Results

| Results of static analysis | Block 1 | Block 2 | Block 3 |
|---|---|---|---|
| Missing asynchronous reset synchronizer | | √ | |
| Unexpected gate in reset tree | √ | √ | |
| Reset signal used as asynchronous and synchronous | | | √ |
| Good asynchronous reset synchronizer | | √ | |

| Number of RDCs | Block 1 | Block 2 | Block 3 |
|---|---|---|---|
| Across same clock domain | 3 | 25966 | 225 |
| Across different clock domains | 0 | 2618 | 42 |

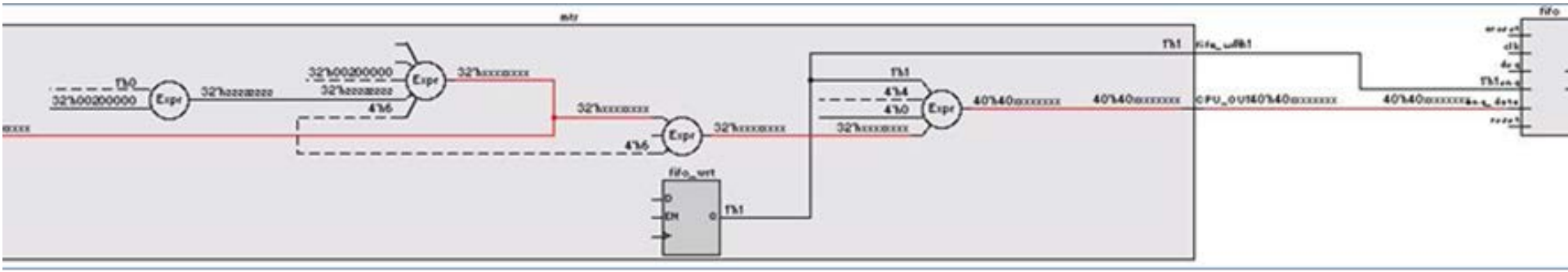
# Sample RDC Violation



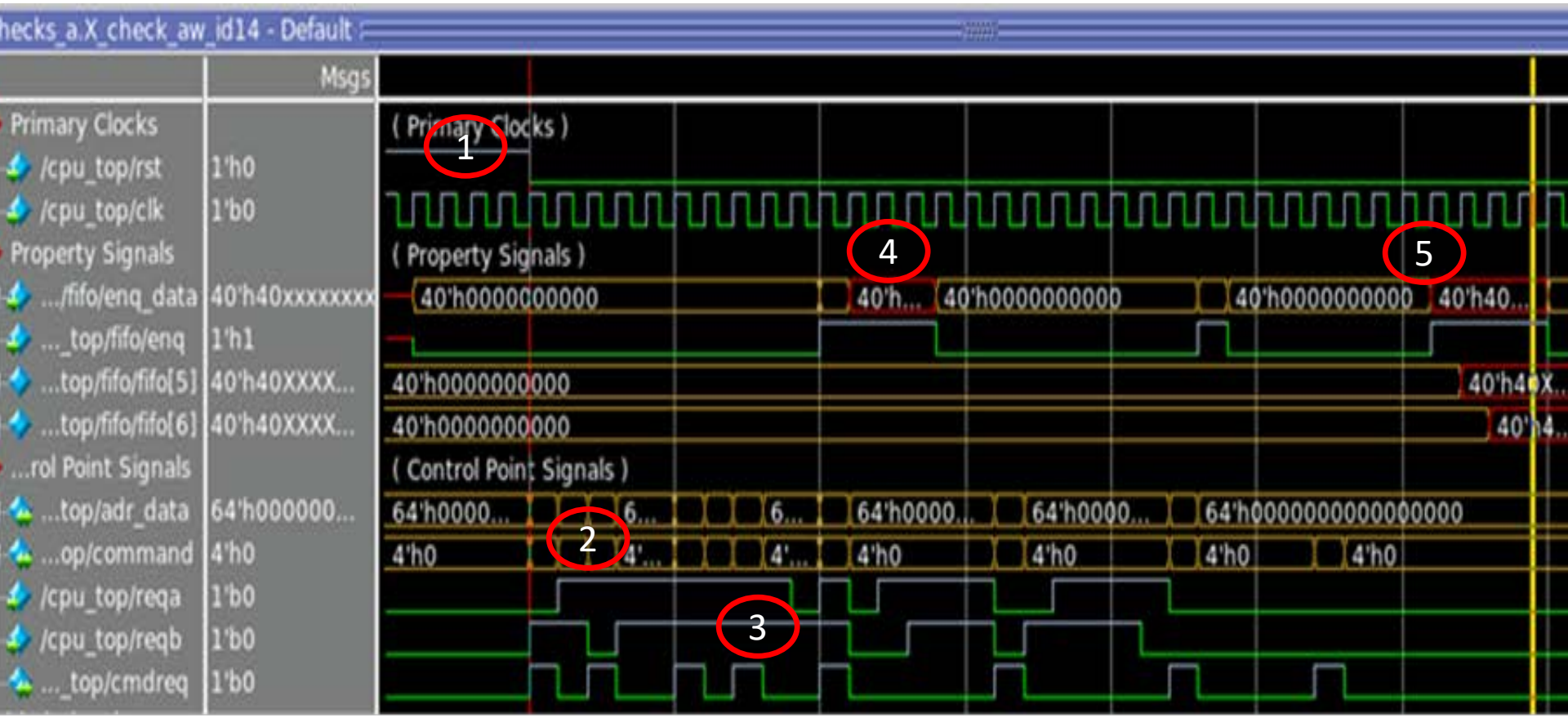

- Signals “mask” and “rx_masked_data” are in the same clock domain
- “rst” and “clr” are of different reset domain

# Formal Results

- To validate a FIFO is functioning properly
- Questa Formal was used



# Formal Results



# Summary

- As SoC designs get complex, reset problems become harder to catch
  - Traditional solution no longer sufficient
- We discussed:
  - Several common reset problems presented
  - Proposed solution discussed
  - Using the method, results were presented