Exhaustive Reset Verification Enablement: Client PCIE Design Reset Verification Case Study

Rohit Kumar Sinha, (rohit.kumar.sinha@intel.com)

Praveen Dornala, (praveen.dornala@intel.com)

Abstract

Traditionally reset design and verification are done by design engineers' and they are expected to follow some standard reset architecture guidelines to avoid any potential meta-stability issues. However, with the advent of complex power management design flows and due to the increase in reset signaling complexity with the emergence of multiple reset domains, reset domain crossing verification becomes an absolute need to ensure glitch free reset assertions during various power states. Also, client SOC designs contain a high level of complexity in the reset distribution and synchronization circuitry and verifying that a design can be correctly reset under all modes of operation presents a significant challenge that can't be addressed in RTL simulations. Therefore, there is a tangible need to provide automated, exhaustive structural and functional reset signaling checks. This paper details about exhaustive reset verification in the PCIE Subsystem and provides a detailed coverage of all the metastability and reconvergence issues, reset domain crossing issues that could lead to silicon bugs while improving the quality of handoff to the backend team

***Keywords*—** reset, meta-stability, verification, asynchronous, reset domain crossing, reset tree, Client, SoC

I. INTRODUCTION

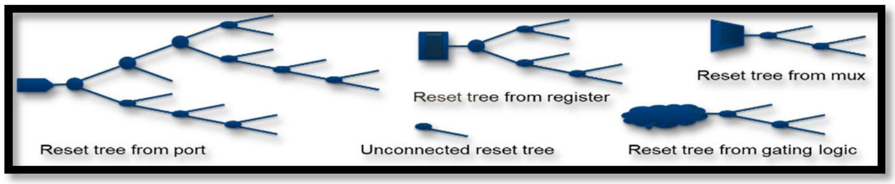
The purpose of this paper is to describe the methodology with a detailed step by step approach to handle common reset verification challenges, and to discuss how the problems can be identified through a variety of different techniques using exhaustive reset verification. Below is the comparison of reset coverage in the existing Client methodology with respect to the proposed Reset Verification Tool

In the section, we highlight some of the common problems and challenges identified in PCIE subsystem related to reset verification. We will separate it into two main categories:

- Issues related reset distribution tree
- Issues related to the reset usages. In the reset tree section, we focus on whether the reset tree was implemented correctly or not. In the reset usage section we focus on once reset is correct, whether the usage is correct or not

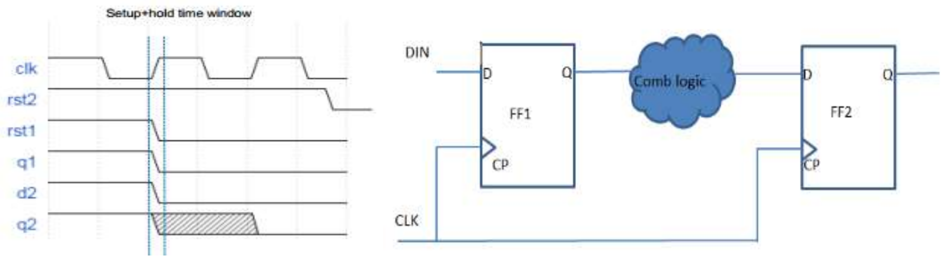
A. *Reset Distribution Tree*

The first category of reset problems in design is whether reset was implemented incorrectly. The problem usually shows in the reset tree. Figure below shows an example reset tree. If the reset tree is incorrect, then all the other checking will be incorrect, and the chip will not function properly. There is a long list of potential problems that we have seen, and we highlight 2 common problems here.



III. RDC PROBLEM

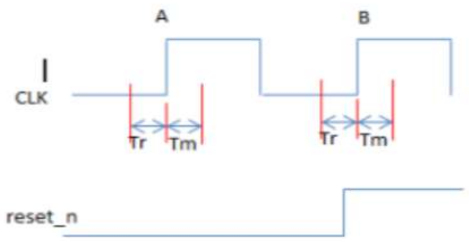
The asynchronous assertion of the reset in the launch flop can occur close to the clock edge of the both the flops, the launch flop will any way enter into reset assert condition but the problem is with the capture flop which enters into metastability due to sampling asynchronous data change due to asynchronous reset.



A. *Reset recovery and reset removal*

Async reset DE-ASSERTION must be timed – recovery/removal checks w.r.t clock pin of the async reset flop. Recovery constraint specifies the minimum time that an asynchronous control input pin must be held stable after being de-asserted before the next clock (active-edge) transition. Removal constraint specifies the minimum time that an asynchronous control input pin must be held stable before being de-asserted after the previous clock (active-edge) transition.

Recovery time is like the setup check, in that this is the time the asynchronous input should be stable before the arrival of the clock. Similarly removal is the equivalent of hold check, in that the asynchronous input should be held after the clock edge. In the below figure Tr is the reset recovery time, and the Tm is the reset removal time.



B. *Resets used with mixed (asynchronous/synchronous) types*

In general reset can only defined as asynchronous or synchronous to a clock. Sometimes, it’s being used as synchronous reset for clock1, and asynchronous for clock2. This usually indicates a misunderstanding of the reset in the system. The following RTL illustrates a clock being used as an asynchronous reset on register q1 and a synchronous reset on register q2.

C. *Flip-flops with overlapping set and reset.*

It is generally not a good idea to allow both asynchronous reset and set exist in a flip flop and is not common in synchronous designs. In fact, many technology libraries do not provide a flip flop that has both asynchronous reset and set. This problem is usually only a simulation problem and not synthesis issue. The RTL below may not simulate correctly because of the behavior of the simulation model. When both set and reset are both active asynchronously, the verilog always block may miss one of the reset or set states leading to race condition and there will be a mismatch between simulation and synthesis. An RTL example of overlapping set and reset is shown below.

If set_n and rst_n are asserted together, there is a race condition. The resulting behavior of this register will depend on the physical implementation of this cell. One solution to avoid set and reset race conditions is to convert one of the asynchronous set or reset signals into a synchronous signal. The following RTL avoids a race condition by using an asynchronous reset and a synchronous set.

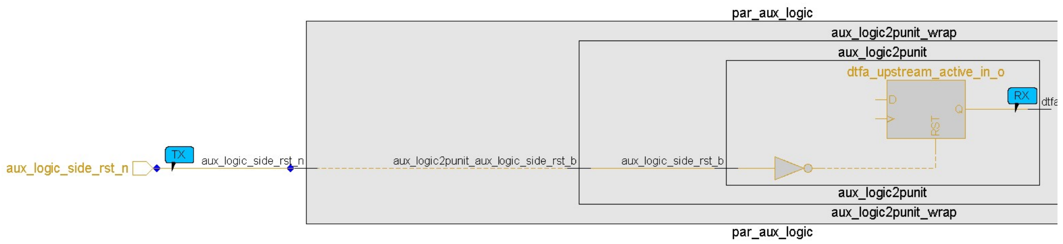
IV. RESET USAGE

In the category, the application of reset and its related issues are discussed. To understand it we need to first define the terms reset domain and clock domains. A reset domain is characterized by the following attributes – a) Type – synchronous or asynchronous b) Polarity – active low or active high c) Value – set for 1 and reset for 0 and d) the top resetting signal

In a design with multiple reset domains and clock domains, it is important to make sure resets are used properly. The reset tree annotates the sequential elements in the design with the reset domain information. With every register in the design having a clock domain and reset domain – the usage of the resets in a particular clock domain can be determined. With this information, we can then identify asynchronous reset crossings across sequential elements in the same clock domain, as well as identify crossings between asynchronous and synchronous resets across registers in the same clock domain. These paths between resets are called Reset Domain Crossing (RDC) paths

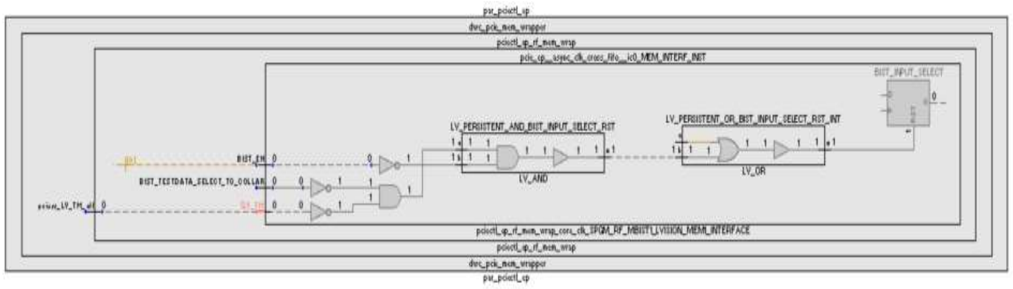
A. *Missing synchronizer for asynchronous reset*

All asynchronous reset signals should be synchronized to the target clock domains before used. The problem is not the assertion of reset but with the de-assertion (release) of the reset. If the reset is de-asserted close to the next clock edge – it may violate the reset recovery time leading to metastability. When an asynchronous reset violates the setup and hold requirements for a register, this violation is likely to happen simultaneously on multiple registers and the register metastability will result in a random delayed release of reset on the metastable registers.



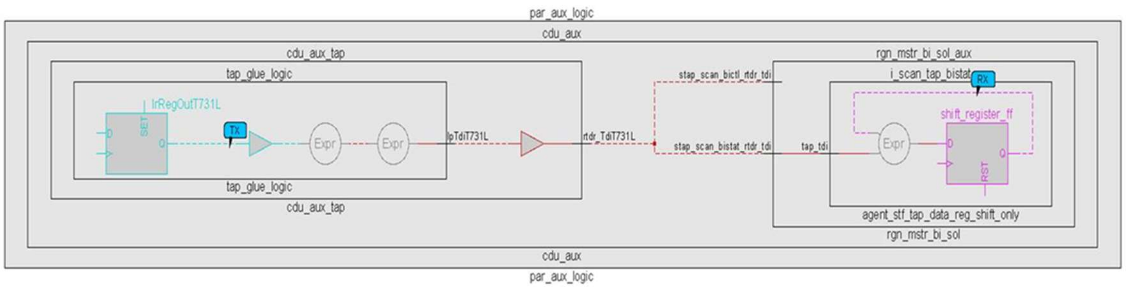
B. *Polarity Correctness and Reset Constant Propagation*

It is important to check the reset correctness with the design intent. Two common problems are that resets are used with wrong polarity or clock. Figure 5a shows that the downstream register using reset synchronizer is clocked in a different clock (CLK2) from reset synchronizer (CLK). CDC analysis [5] will detect a CDC crossing and the problem would be identified during CDC verification. Figure 5b shows that the register reset by the synchronizer is using active high reset, while the synchronizer is an active-low reset. Since this is not a CDC crossing, it will not be caught by CDC tool, and cannot be caught easily. Figure 5c shows that the constant is being propagated and making the reset always asserted.


par_pciectl_op
dwc_pcie_mem_wrapper
pciectl_op_rf_mem_wrap
LV_PERSISTENT_AND_BIST_INPUT_SELECT_RST
LV_PERSISTENT_OR_BIST_INPUT_SELECT_RST_INT
BIST_INPUT_SELECT
LV_AND
LV_OR
BIST_EN
BIST_TESTDATA_SELECT_TO_COLLAR

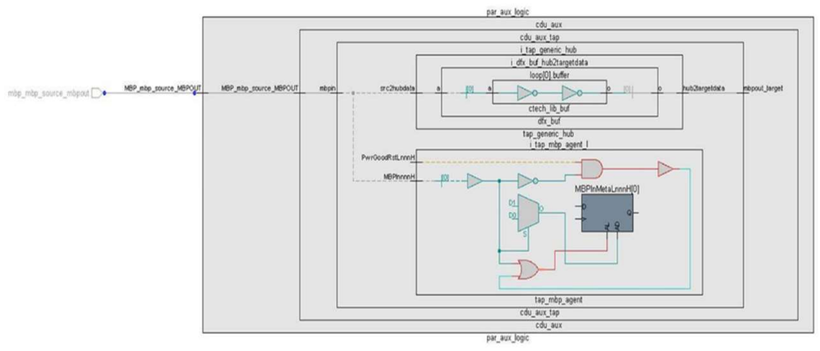
C. *Reset Domain Crossing*

This crossing of reset signals from one reset domain to another reset domains cause metastability risk. If RST1 is asserted while RST2 is not asserted, DFF2 can sample asynchronous data. If the data transition is within the setup and hold window, then the DFF2 output will become metastable. Below pictures shows two different kinds of crossing. First case is set to reset crossing and second being reset to reset crossing.



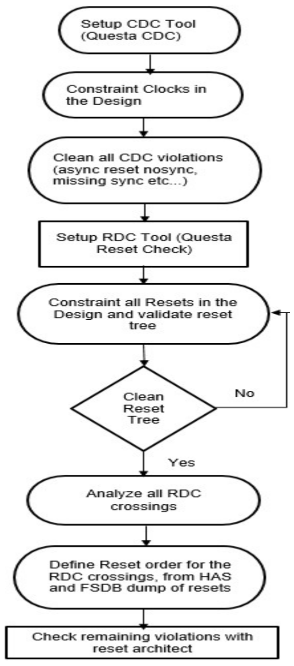
D. *Reset Convergence*

Another aspects of reset application is reset convergence wherein reset signals splits into paths that converge at a signal that gets used as async reset as shown in the picture below



V. RESET VERIFICATION FLOW

Flow of process of RDC Verification


Setup CDC Tool (Questa CDC)
Constraint Clocks in the Design
Clean all CDC violations (async reset nosync, missing sync etc...)
Setup RDC Tool (Questa Reset Check)
Constraint all Resets in the Design and validate reset tree
Clean Reset Tree
No
Yes
Analyze all RDC crossings
Define Reset order for the RDC crossings, from HAS and FSDB dump of resets
Check remaining violations with reset architect

VI. RESET ANALYSIS

A. *Reset crossing reported*

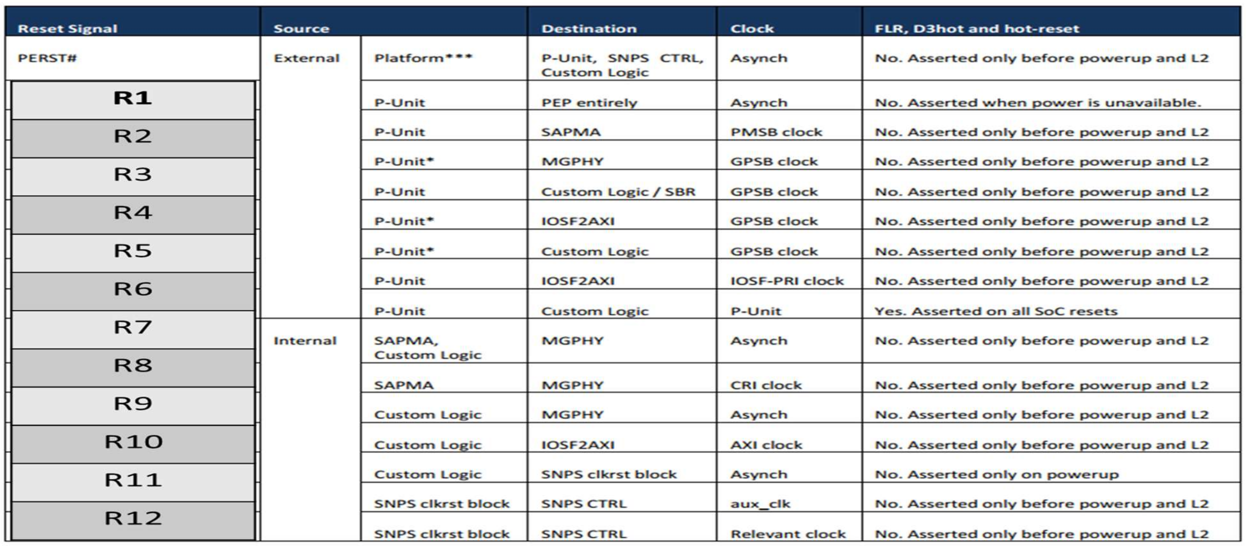
There are more than 20 reset groups and there are essentially 10+ reset domain crossing which leads to 10000+ reset crossings. It is very essential to provide the reset to define the reset ordering in order to assertion and desertion sequences.

B. *Reset assertion order*

Many of the RDC crossings can be resolved, by knowing the reset sequence of assertion. In the RDC crossings if the DFF2 (Rx sampling Flop) is asserted well before the assertion of the DFF1 (Tx Flop), then the asynchronous changes in the data will not be sampled by the Rx Flop as it is in reset assertion state.

C. *Reset Ordering from High Level Architecture Definition*

From the architecture specifications (HAS) and the FSDB load of the simulation in the Verdi has helped us in finding the Reset assertion sequence. Below is the architecture specifications of the reset signals in the subsystem, from this many of the resets are asserted before power up.



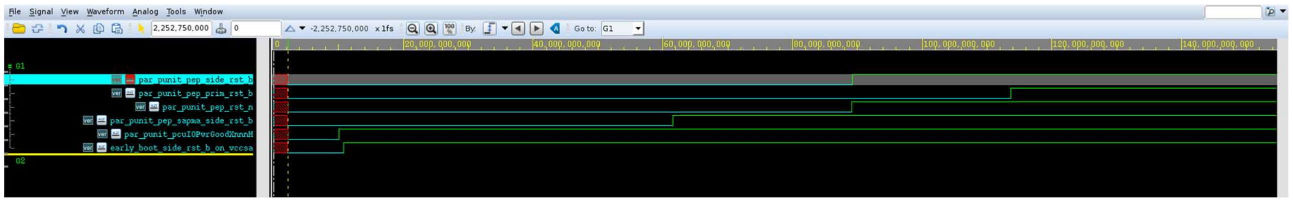
Reset Signal	Source		Destination	Clock	FLR, D3hot and hot-reset
PERST#	External	Platform***	P-Unit, SNPS CTRL, Custom Logic	Asynch	No. Asserted only before powerup and L2
R1		P-Unit	PEP entirely	Asynch	No. Asserted when power is unavailable.
R2		P-Unit	SAPMA	PMSB clock	No. Asserted only before powerup and L2
R3		P-Unit*	MGPHY	GPSB clock	No. Asserted only before powerup and L2
		P-Unit	Custom Logic / SBR	GPSB clock	No. Asserted only before powerup and L2
R4		P-Unit*	IOSF2AXI	GPSB clock	No. Asserted only before powerup and L2
R5		P-Unit*	Custom Logic	GPSB clock	No. Asserted only before powerup and L2
R6		P-Unit	IOSF2AXI	IOSF-PRI clock	No. Asserted only before powerup and L2
		P-Unit	Custom Logic	P-Unit	Yes. Asserted on all SoC resets
R7	Internal	SAPMA, Custom Logic	MGPHY	Asynch	No. Asserted only before powerup and L2
R8		SAPMA	MGPHY	CRI clock	No. Asserted only before powerup and L2
R9		Custom Logic	MGPHY	Asynch	No. Asserted only before powerup and L2
R10		Custom Logic	IOSF2AXI	AXI clock	No. Asserted only before powerup and L2
R11		Custom Logic	SNPS clkrst block	Asynch	No. Asserted only on powerup
R12		SNPS clkrst block	SNPS CTRL	aux_clk	No. Asserted only before powerup and L2
		SNPS clkrst block	SNPS CTRL	Relevant clock	No. Asserted only before powerup and L2

D. *Mapping of Pciess resets with SOC*

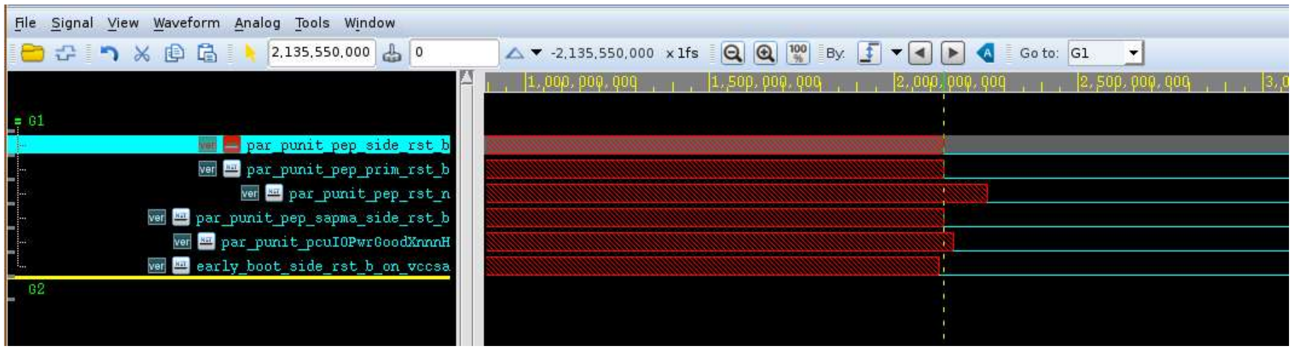
Below is the Pciess reset signals and its driver signals in the soc. All the Pciess EP reset signals are mainly driven from the Punit at Soc level. The mapping between the Punit signals and the Pciess EP reset signals is shown in the below table.

Pciess Resets	Soc Resets
R1	SoC R1_1
R2	SOC R2_1
R3	SOC R3_1
R4	SOC R4_1
R5	SOC R5_1
R6	SOC R6_1
R7	SOC R7_1

Below is the FSDB dump of the signals of pciess resets at SOC level with their driver resets signals from Punit.


File Signal View Waveform Analog Tools Window
2,252,750,000
G1
par_punit_pep_side_rst_b
par_punit_pep_prim_rst_b
par_punit_pep_rst_n
par_punit_pep_sapma_side_rst_b
par_punit_pcuIOPwrGoodXnnnH
early_boot_side_rst_b_on_vccsa
G2

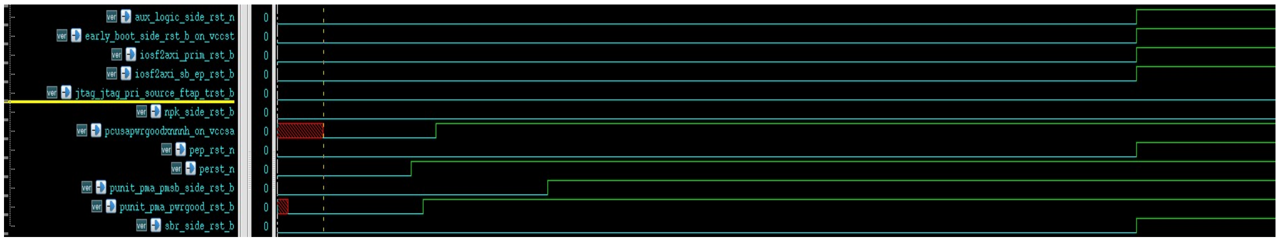
By looking at the order in which the resets got asserted, first is the assertion of early_boot_side_reset_b followed by prim_rst_b and side_rst_b, sapma_side_rst_b are asserting at the same time, and then the pcuIOPwrGoodXnnnH got asserted and lastly the pep_rst_n.

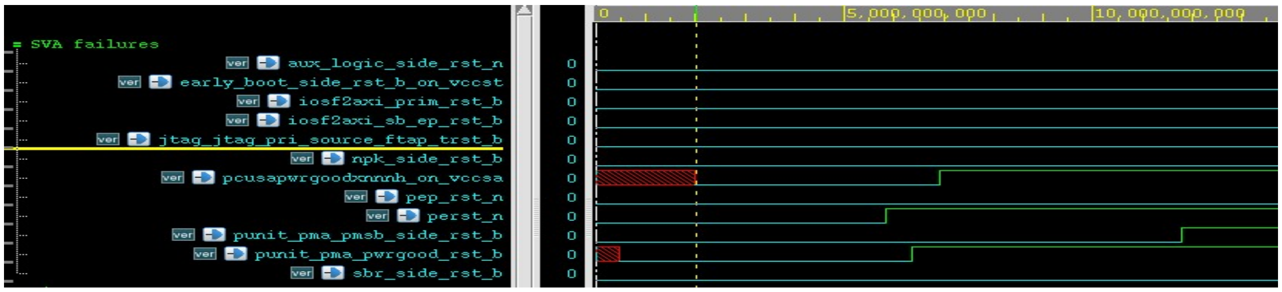


So any crossing reported with Pep_rst_n as the TX reset and any of the above signals as Rx reset, the crossing is safe as the Pep_rst_n is the last one to assert and any other reset (resetting the Rx Flop) will already be in reset assertion state and blocks the asynchronous changes

E. *Mapping of Pciess resets with SOC*

Below is the loaded FSDB of pciess Ep Subsystem in the Verdi. Here are the waveforms of the all the primary resets signals with the assertions and de-assertions shown.





If we consider the above RDC crossings, all the Rx resets from the FSDB dump of both Subsystem and Soc and HAS are being asserted before the power up. The TX reset punit_pma_pwrgood_rst_b from the waveform is being asserted after the assertion of the all the above RX resets, there by avoid metastability to propagate to the RX Flop.

F. *Reset Order in RDC Tool*

In Reset check tool, the assertion order of these resets can be given by the following directive.

```
resetcheck order assert -from R1 -to R2
resetcheck order assert -from R3 -to R4
```

G. *Clock gating*

The other solution to RDC is to gate the clock. By shutting of the clock for a few cycles when there could be a meta-stability due to resets being asserted, we can avoid meta-stability issues. This technique is very elegant, but comes with many side effects. This is not always possible and is generally not used at a large scale level

VII. RESULTS

We ran exhaustive reset verification using Reset Verification Tool on PCIE End Point Subsystem consisting of 5 IPs of varying complexity including a functional controller, PHY block, side band router, DFT block etc. Each of the designs show different interactions between the generated clock domains and the inferred reset domains leading to domain crossing issues. In general, users can customize the analysis by assigning explicit values to the control signals, or by grouping the clock and reset signals explicitly with directives. PCIE End Point Subsystem Design has 24 reset domains and 20 different clock domains.

There were many design issues as mentioned below that got uncovered by enabling exhaustive reset verification.

- User specified resets were unused
- Mismatches between architectural specification and reset usage
- Combinational logic before asynchronous reset synchronizer

REFERENCES

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

[2] Questa ResetCheck, https://www.mentor.com/products/fv/questa-reset-check

[3] Dealing with SoC metastability problems due to Reset Domain Crossing by Arjun Pal Chowdhury , Neha Agarwal and Ankush Sethi, Freescale Semiconductor India Pvt LTD

[4] Multi-Domain Verification: When Clock, Power and Reset Domains Collide by Ping Yeung, Erich Marschner