Reset Sweep Verification for Elimination of Reset Domain Crossing Blind Spots in Design

Nitika Gupta, Neha Srivastava, Vivek Yadav
NXP Semicondcutors
Email:- nitika.gupta@nxp.com, neha.srivastava@nxp.com, vivek.yadav@nxp.com

Abstract

This paper presents a novel verification strategy, targeted to stress design and architectures having multiple reset and clock domains with cross-interactions between these domains, to intercept functional limitations of design earlier. The methodology randomizes valid attributes of bus initiators and targets and excites combinations of interconnect traffic. Further, it asserts granular, incremental reset events around active use-case traffic to catch system hangs, glitches, transaction losses and other such functional impact of reset domain crossings, which we will refer to as functional RDC in paper. This methodology is applied on a state-of-the-art automotive processor chip for Software Defined Vehicle (SDV)**.** The methodology iteratively provides feedback to architecture and design from day one of pre-silicon definition and RTL development while augmenting traditional structural methodologies which are increasingly insufficient for comprehensive signoffs. By intercepting possible anomalous behaviors in pre-silicon verification, quality of the chip is made robust. Complex silicon bugs and their tricky, expensive debugs can be eliminated during in-field operation.

Index Terms

Reset Domain Crossing (RDC), Bus Quiescing, Bus Draining, System on Chip (SoC), Traffic profile, Stress Verification, Functional RDC, Reset Sweep Verification, Software Defined Vehicle (SDV)

I. INTRODUCTION

A. Background

In a typical modern system on chip (SoC), to sustain various use-cases, logic is very commonly distributed into multiple reset domains, designed to cater different functional paths and behavior requirements. These reset domains are further expanding due to multiple software configurable reset which resets partial logic in design to support the domain and zonal architectures of Software Defined Vehicles. Hardware faults can occur as ECC/parity, supply and clock tree misbehaviors, processor malfunction, manufacturing anomalies etc. For safety critical applications, reactions of these faults are typically accompanied by plurality of asynchronous resets in chip. Additionally, there can be intentional reset events like pad resets or debugger-initiated system resets.

There is potential risk of losing 'in-flight' data or even create system hangs or glitchy interactions between the reset domains when partial resets are asserted, and entire system is not taken through reset. In such cases, initiator agents may attempt to issue a transaction to target agent that might be in reset. This can result in initiator component receiving erroneous response or hang states leading SoC into undesirable states. Alternatively, if initiators are reset while targets stay active on higher reset domain, bus transactions which are abruptly terminated can lead to spurious writes on control logic or faulty status bit setting. This may further lead to SoC misbehaviors including reset escalations reducing device availability. In Fig.1, a complex model of Reset Domain Crossing (RDC) which is easy to comprehend the magnitude of potential bugs due to RDC issues across design is illustrated.

While traditional RDC tools & structural techniques can extract structural lists of such crossing points, the scope of analysis is in order of 100s of 1000s of violations. There is an overall industry gap in methodologies which can narrow down or intercept functional manifestations of such crossings. Analyzing and disposing of these violations is still a heavily manual process and hence very prone to bugs [1]. Even if structural RDC pass, transactions can get lost or corrupted if either initiator/target is reset before its completion. This is a functional and not structural issue. The EDA tools are not scoped for intercepting such functional RDC issues. Structural RDC can give a vast, exhaustive list of reset domain crossing points but the methodology which we describe in this paper is still needed to excite, visualize, and intercept the functional implications of reset domain crossing.

B. Existing static RDC check is not enough?

RDCs cause chip failures which require silicon re-spins. Below are some points which highlights the voids in conventional RDC verification, especially on SoC. Hence it becomes crucial to ensure the design assumptions around dispositioning RDC points are correct through robust RDC verification methodology for accurate and efficient analysis. This includes testing protocol equivalence, design assumptions, constraints and reducing the false reporting of RDC's [2] [3].

- Improper constraints, assumptions, and configurations in RDC analysis can lead to 1000s violations which are redundant noise in the static analysis. For false violations verification engineers apply waivers. This adds to another effort intensive task of waiver management. Further, it highlights the limitation of the human predictability to waive off blind spots cross scenarios in design which might be a corner scenario and cannot be caught by tool.

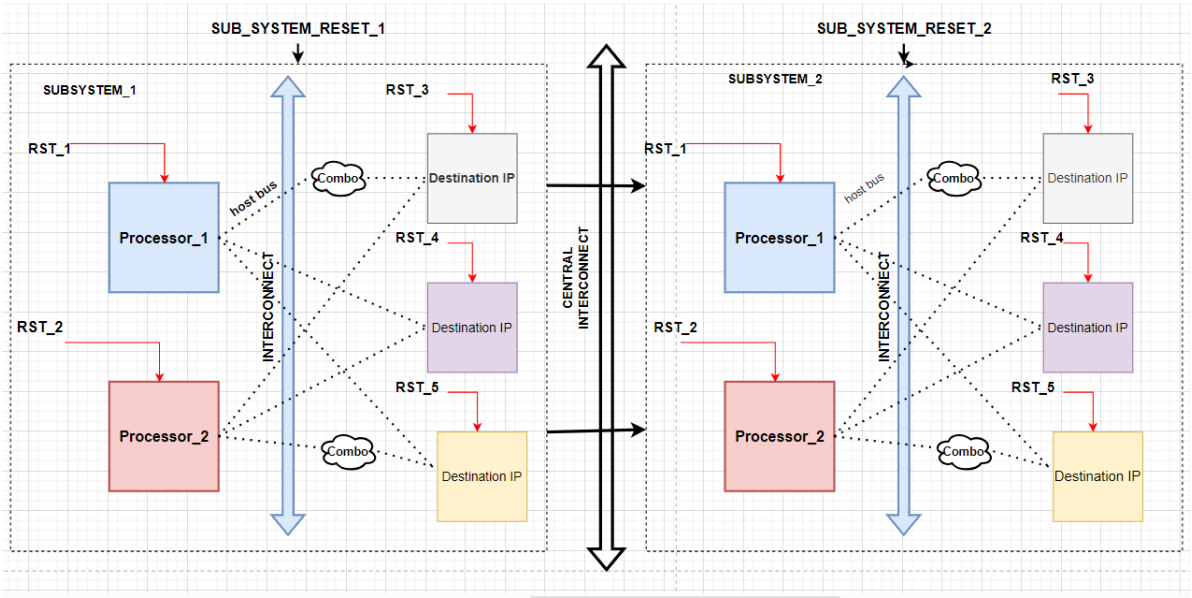

Fig. 1. Illustrating complex RDC paths crossing in SoC having multiple subsystems with plurality of resets

- Designer might categorize the RDC properly but there are chances of missing design issues across dynamic behavior of chip during different modes of operation. It is coveted that the tools and RDC verification flows should have an innovative way which has an automated data analysis or machine learning solution which can detect the non-trivial issues in the setup and give corrective suggestions to verification engineers to eliminate re-spins or re-synth due to RDC bugs defects.
- Static checks do not account for timing. These checks may confirm the presence of RDC mitigation logic but do not analyze whether the timing between domains is safe, or if there are metastability risks under certain operating conditions.
- Modern systems often have complex reset architectures, including multiple levels of resets on single Block or IP in SoC (e.g., soft resets, hard resets), and intricate reset control mechanisms. Static checks might not fully capture or verify the proper sequencing and dependencies in these complex systems. Risk grows exponentially as design magnitude, complexity and reset convolution increases.

C. Functional RDC verification as emerging need :- Illustrative Example

In most SoC's, an external debugger can be attached to chip to investigate the functionality of chip post silicon and these intrusive debug transactions can continue even under the lower grades of reset assertions. IO peripherals and memories are integrated in design for several purposes. To connect these elements, a transport network is provided over which data, which may be in the form of packets, can be transferred from a source element to a destination element. Each element may have a different reset domain associated.

Consider an architecture where interconnect lying in data access path is connected to a reset which is at a lower grade than reset of debugger. So, there can be cases where debug transaction is initiated to the SoC elements under reset or reset might occur in chip while a debug operation was on-going (Fig. 2). In such instance, there might be unexpected packet loss or debugger (or other requesting processors) might go into a deadlock waiting for response from an unavailable target of transaction or a protocol violation might occur which corrupts chip operation overall. There exists a need for a mechanism to avoid such real time undesirable scenarios on silicon and exhaustive verification to drive correct design and architecture early in cycle through development stage feedback.

It is desirable that the debugger receives responses for the issued transactions gracefully before the target agents or operating path goes into reset. In case this is not achieved, the only route left to avoid corruption of retained logic is to assert the reset to the debugger but with penalty of loss of debugging functionality, system availability and safety diagnostics. Design would lose the potential to support a scenario where the debugger itself is trying to debug the challenging state. If user had to debug some anomaly in the functional reset scenario (safety reset reaction, security core errors) using debugger utility, they would have wanted details of failure reasons and exhaustive status bits. A reset which clears the debugger is undesirable in this case as it would clear the evidence of the cause of reset inside chip under debug mode.

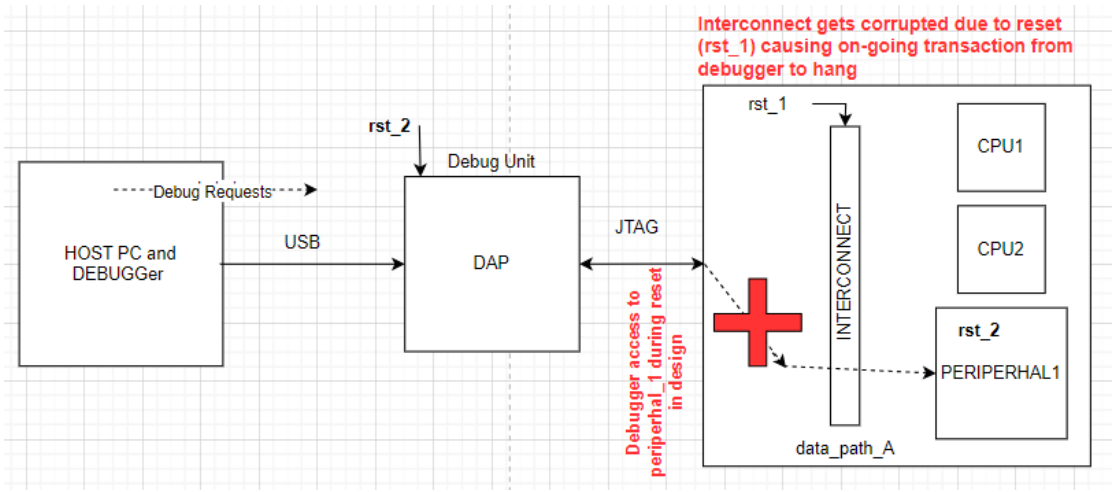

Fig. 2. Example of Silicon bug missed by RDC tool caught in post-Si Validation.

II. ENHANCED ARCHITECTURE & DESIGN

A. Bus Quiescing and Draining (BQD)

Quiescing is the process of isolating a node of a cluster or protecting shared resources when a node appears to be malfunctioning. This is general concept of quiescing in system application. It is widely implemented in computer applications to manage system applications on software. In the same context, it is applied to RTL where peripherals are being isolated from processors during unexpected malfunctioning activity in design. In safety-critical applications, safety-critical bus host, such as the application cores, are implemented redundantly and operate in lockstep mode to improve the error prediction mechanism and reliability of the system. Upon detection of a lockstep mismatch, the recovery mechanism ideally only involves restarting the lockstep unit while rest of the system operates un-interruptedly. The gasket provides a means to gracefully isolate the safety-critical source from the rest of the system. The gaskets accomplish this by: -

- Quiescing off the initiator such that any new requests issued by the processor or initiator are not propagated downstream to the interconnect. Additionally, an error response may be sent back to the initiator.
- Draining all outstanding transactions previously issued by the processor or initiator. This involves monitoring all incoming responses and calculating when all expected responses have been received [4] [5]. Completing all unfinished write requests on behalf of the initiator, which is now presumably unreliable, as indicated by the lockstep fault.

A mechanism (Fig. 3) is needed to gracefully manage the on-going transaction on interconnect from initiator to target and quiesce the in-flight stream of packets from initiator to target agent till reset phase is completed in design.

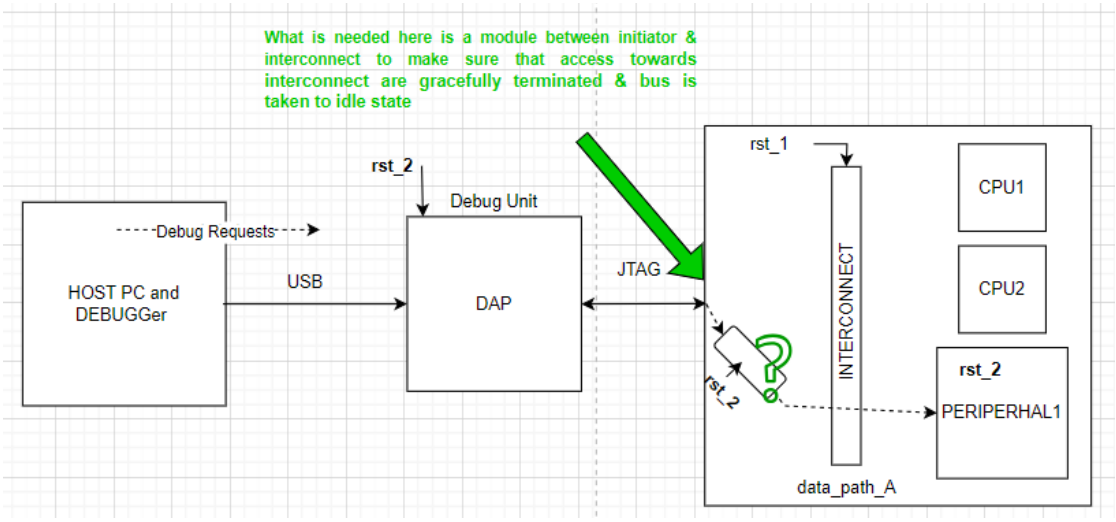

Fig. 3. Debugger path for accessing logic sitting in different reset domains.

B. Architecture and Design

It is proposed to place the Bus Quiescing and Draining logic (BQD) gaskets across the RDC's path where the source, destination and operating path has multi-reset domain paths crossing (Fig.4). Before propagating the reset in design, an early request can be triggered to the BQD gaskets which should gracefully complete on-going transaction if any on interconnect and operating path from source to destination and block further transaction from source to destination via asserting isolations in design. Once this process is completed, an acknowledgement can be traversed back to the reset generation IP which will hold the propagation of reset till it gets acknowledgment back from system. The requesting agent may continue to issue transactions, but bus quiescing and drain will start responding on behalf of target agent while operating interconnect or target agent is still in reset.

The placement and number of this gasket depends on reset plurality and complexity of the design. With this assumption that a complex design might have hardware and software reset domains the integration of BQD would be aligned to it. Even in scenarios (in an SDV architecture) where reset could be asserted by SW configurability, BQD is useful to initiate a graceful traffic quiescing before reset is propagated through software. Additionally, as test and debug hooks, there should be mechanisms in place to override or timeout such handshakes if design genuinely gets hung on real silicon executing the Bus Quiescing and idling.

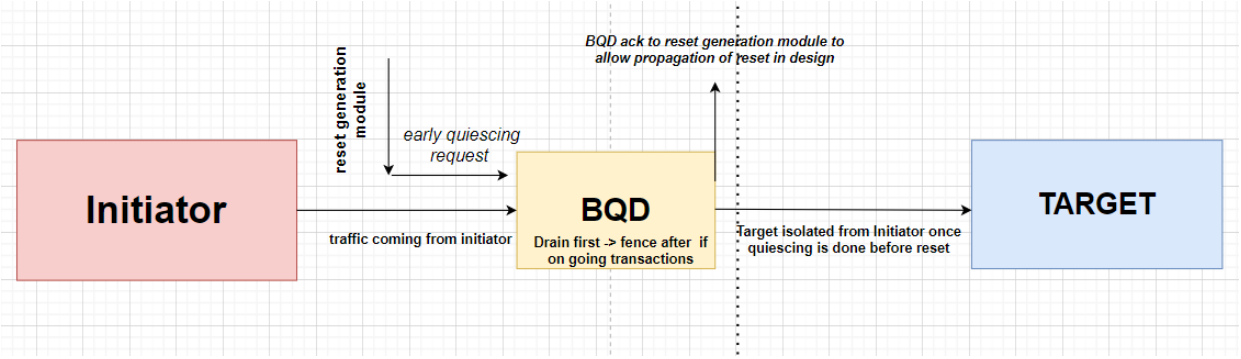

Fig. 4. Bus Quiescing and Draining sequence.

III. FOCUSED RDC VERIFICATION METHODOLOGY

A. Design Verification Model

- Traffic generation through actual initiators or targets for verification can become un-controllable. A general interconnect Bus Functional Model (BFM) or traffic profile driver to generate bus traffic with randomized initiator attributes is integrated in Testbench to mimic the processor. These drivers issue the random transactions (to cover exhaustive combinations of initiator /target protocol attributes) to the target agent across an interconnect while hardware reset events are generated granularly and incrementally over cycles to sweep asynchronous resets over in-flight traffic (Fig 5).
- This scheme is more robust in catching the blind spots in design which ends up as silicon bug. It is quite scalable and portable approach complementing static RDC checks in a Multi-core Multi Reset with partial system reset.
- In the testbench environment, we also have random delays modeled to create delays between asynchronous components even in RTL verification to mimic timing, including the asynchronous logic between processor vs interconnect and interconnect vs target. This randomization is also stressed to further intercept corner scenarios.
- Quiescent Formal Checks [5] or Direct System Verilog assertions [6] are written on the overall AMBA protocols as well as reset state machines and handshake signals of design to highlight micro-architecture gaps and Integration issues.
- Protocol checkers/monitors are bound at boundary of traffic profilers to catch protocol violation/loss during reset at buses.

B. Design Verification Environment

The verification flow as shown in Figure 6 and Figure 7 uses a UVM sequence which stitches the random traffic generation using the testbench integrated traffic profilers and in parallel generates the random reset sweeps with random delay. The sequence is written in such a way which can be controlled via switches to increase the number of transactions or tweak the delay added for each asynchronous reset. An embedded C testcase with cores being put in WFI along with UVM sequence executes in parallel on design.

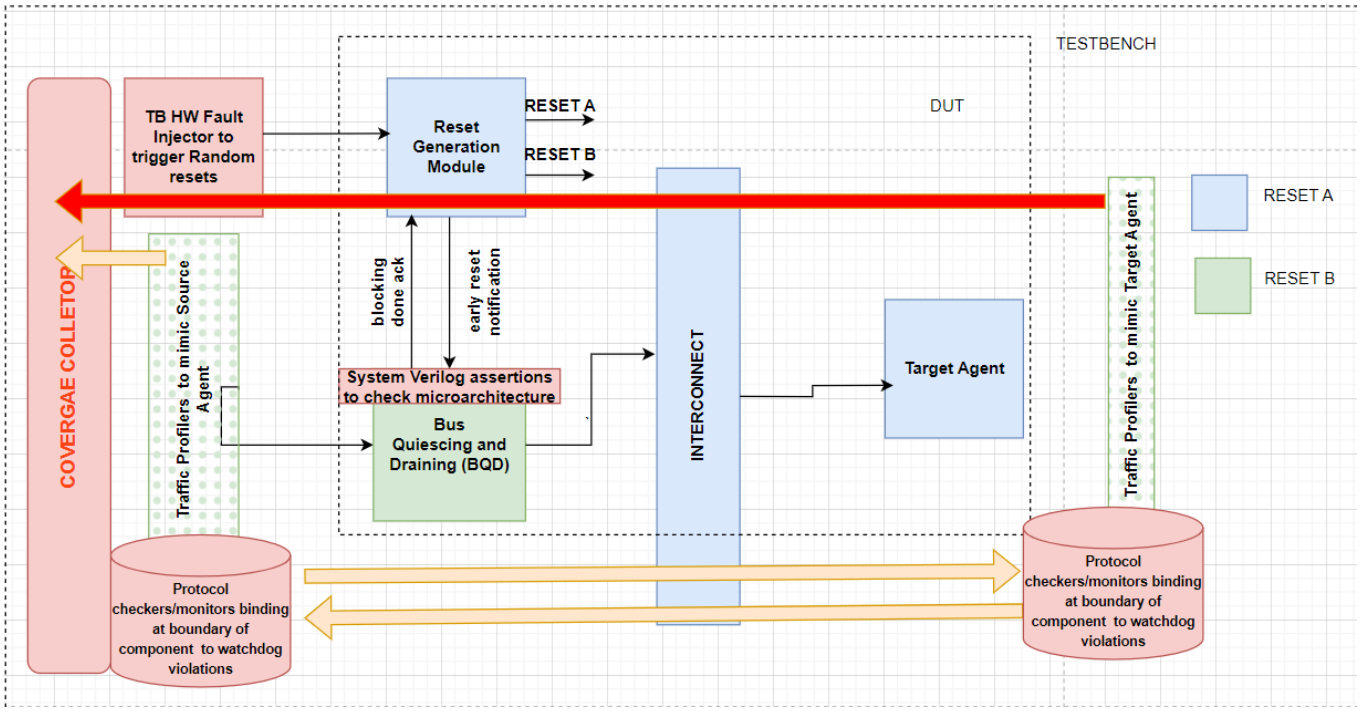

Fig. 5. Illustrating Design and Verification methodology

```
class child_sequence extends base_sequence;
    `uvm_object_utils(child_sequence)
    int kk,ii;
    int transaction_size = 0;
    int no_of_transactions;
    int transaction_type;
    int random_us;
    int random_ns;

    /** Class Constructor */
    function new (string name = "child_sequence");
        super.new(name);
    endfunction : new

    /** Task to add random delay **/

    task delay_random_us(input int min, input int max, input string des);
        if($test$plusargs("MAX_DELAY_RANDOM_US")) begin
            random_us = max;
            random_ns = 0;
        end
        else if($test$plusargs("MIN_DELAY_RANDOM_US")) begin
            random_us = min;
            random_ns = 0;
        end
        else begin
            randomize(random_us) with { random_us inside { [min:max] };};
            randomize(random_ns) with { random_ns inside { [0:10000] };};
        end
        `INFO($sformatf("Delay %0d us %0d ns in Proxy model %s", random_us, random_ns, des));
        repeat (random_us) #1us;
        repeat (random_ns) #1ns;
    endtask

    virtual task main();
        int ij;
        super.main();
        if($value$plusargs("NO_OF_TRANSACTIONS=%d",ii))
            no_of_transactions= ii;
         end
        repeat(no_of_transactions) begin //{
        fork
            begin
                    `INFO("Generating Random transctions");
                     generate_random_transaction(addr,transaction_type,transaction_size,0,"",0,0);
                end
            begin
                    delay_random_us(0,20, "Before asserting Functional reset");
                    assert_reset(RST_1); //generate_random_reset
                end
        join
       end
    endtask: main
endclass
```

Fig. 6. Pseudo code for UVM sequence run during simulation

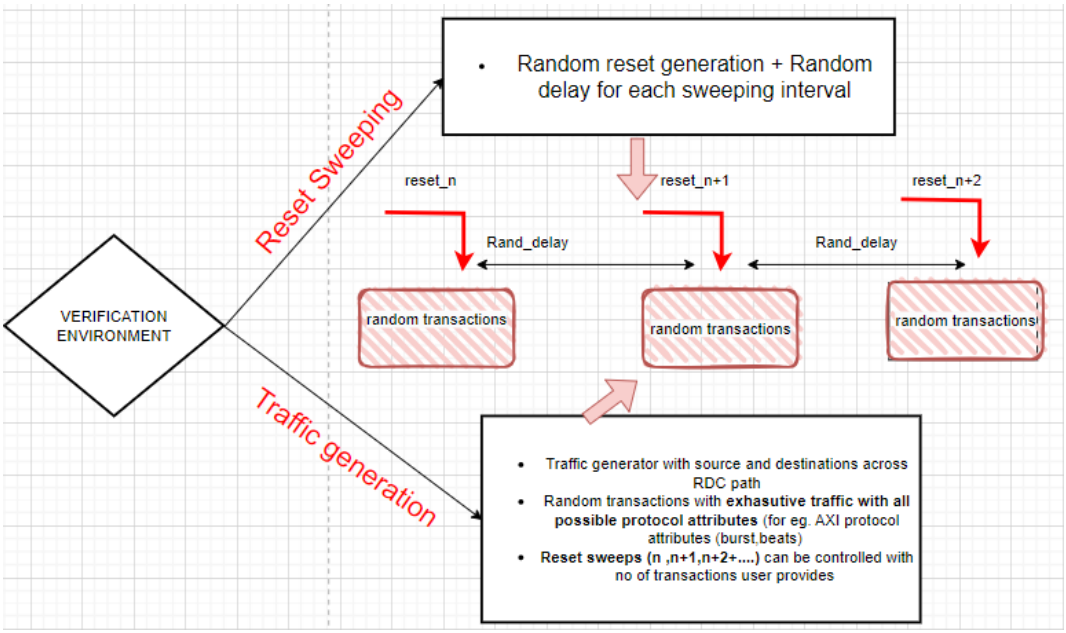

Fig. 7. Flow chart of Verification Environment

IV. RESULTS AND COVERAGE

Applying the random traffic generation and verification techniques described in the previous sections we were able to achieve high coverage of the Reset Domain Crossing paths. Several interesting scenarios were also unearthed that were not thought of during conventional static RDC or directed pattern verification. Examples include dropping of quiescence isolation from debugger path of BQD while device enter and exit from partial reset, back-to-back reset in chip causing hang scenarios etc. Below Table I represents few critical, impactful samples of functional RDC bugs and extreme corner cases which easily escape structural RDC checks but can be easily intercepted early through the methodology (ideally from day 1 of development cycles for IP level designs and SoC integration). Reset sweeping across randomization of traffic parallelly is the backbone of this methodology (Fig. 7).

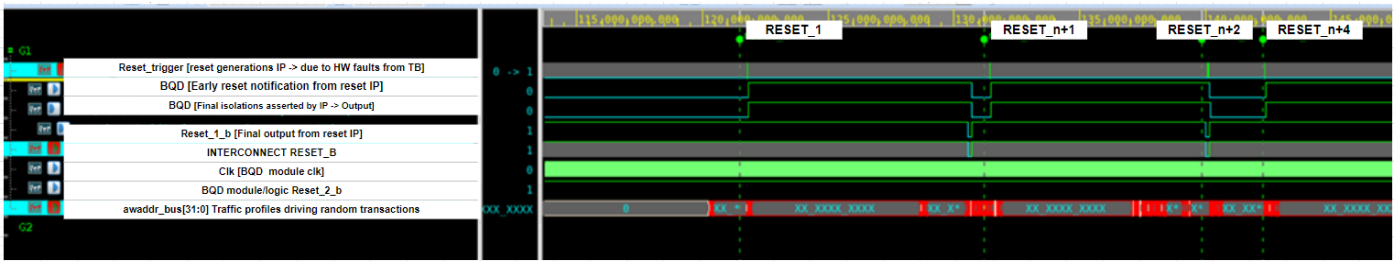

Fig. 8. Simulation results showing random reset sweep across dynamic Traffic

Below Matrix (Fig. 9) shows the sign-off Functional coverage for traffic vs reset condition. RDC reports from EDA tools can be referred as starting point for focus areas by verification engineer to understand combinations of initiator and target RDC paths. Placement of BQD would be across each RDC path (marked with tick). Invalid scenarios are crossed out combinations using structural pre-assessment of design. Verification engineer would add the traffic generator for chosen initiator and target in a testcase and functional scenario of async reset assertion across dynamic traffic is also created further shown by tick marks representing each cycle over which reset is swept. Randomized traffic generation across multiple regressions via UVM sequence gives healthy confidence of all traffic generation scenarios with random reset.

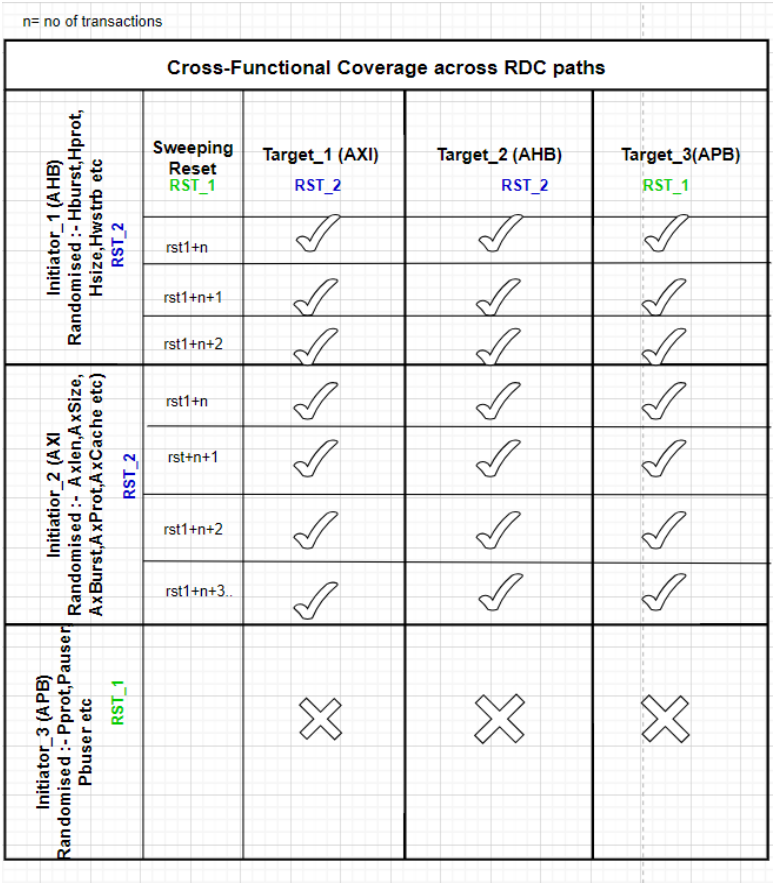

Fig. 9. Functional Coverage Matrix

TABLE I
TYPE OF ISSUES EXPOSED BY RESET SWEEP VERIFICATION METHDOLOGY

Type of Issues	**Design Verification and Severity**	
	Design Issues & Description	**Functional Impact**
1	**Issue :-Back to Back resets propagation is stuck.** Description :- There was hang in design due to BQD resets handshakes were not cleared after reset (wrong reset integration) hence during second reset propagation FSM got stuck.Such issues can be missed if reset sweep verification is not simulated.	SoC Hang
2	**Issue:- BQD module on higher grade resets are dropping BQD request before reset grade 1 is de-asserted.** Description :- This was corner case where due to multiple reset domain BQD which are on higher grade resets were dropping isolation/fencing during assertion of lower reset.	Packet loss due to missing fencing
3	**Issue :- Reset state machine hangs post LBIST as handshake with BQD gasket is not received.** Description :- LBIST assertion did not receive acknowledgement from BQD glue logic due to wrong intergation at SoC level.	SoC Hang
4	**Issue :- When reset grade 1 is asserted , BQD gasket fencing should be high.** Description :- Due to multi-reset domain chip,BQD resets should be in order of respective RDC path reset order however due to wrong integration few RDC in design were dropping the isoaltion signal even during design in reset.	Transaction Loss
5	**Issue :- Debugger doesn't receive response from target when accessed through interconnect during reset grade 1.**	SoC Hang, Debugger protocol violation

Type of Issues	Design Verification and Severity	
	Design Issues & Description	**Functional Impact**
	Description :- Debugger attached during silicon validation was hanging while chip in reset.This was catastrophic bug as only workaround of this bug was to reset the debugger and losing the configuration of debugger to come out of hang.	
6	**Issue:- Target must not transmit the write response before the corresponding address is accepted**. Description :- If transaction packet arrives at the same edge where the isoaltion request from BQD drops, processor gets two responses one from BQD and another from target itself. This was IP bug.	AMBA Protocol Violation
7	**Issue:- Monitor Check for Rvalid held steady until Rready is asserted.** Description :- While BQD was dropping the fencing at completion of reset stage in design , due to bug in BQD gasket there was packet loss from initiator to master on write.There was 2 cycle delay which gasket was taking to drop the isolation after receiving the droping notifcation from system but meanwhile if initiator starts sending traffic request towards target, packet request is lost as BQD is still in progress to drop the request.This was caught from AXI protocol checkers.	AMBA Protocol Violation
8	**Issue:- Monitor Check that the number of write data items matches AWLEN for the corresponding address.** Description: - The initiator issued all beats of write data, including WLAST. In total, master issued N beats of data, but initiator did not issue AW command yet Then BQD isolation is asserted. Expectation is that gasket will issue AW command with *awaddr =reserved_address + awlen=(N-1). Instead, the gasket issued AW command with awaddr=reserved_address + awlen=(**N**)*	AMBA Protocol Violation

V. CONCLUSION

Current state-of-the-art static reset verification methodologies can report reset-domain crossing paths via EDA tools, but it cannot detect and intercept functional RDC failures due to dynamic traffic packet loss across async reset in design or blinds spots like external debugger access during reset in design or protocol violations/packet loss across resets. This paper describes a reset sweep strategy over in-flight bus traffic which is highly scalable to different architecture having multiple reset domains and diverse types of traffic profiles. It saves post-silicon validation debug effort, time, and money on tricky in-field chip failures due to functional RDC weaknesses of architecture and design. The verification strategy is universal and can be reused across Design and testbench environments for efficient and exhaustive signoffs.

This allows multiple random regression cycles across development stages of design thus increasing the confidence of signoffs by making it much more exhaustive and robust. With growing focus on zonal and domain architecture for SDV, Software reset domains are exponentially increasing and the functional RDC complexity would only continue to grow. Hence the methodology which we described in this paper will find increasingly relevant use cases and applications where it helps in early interception of functional RDC bugs. Further work in stitching Machine learning or data modelling in same setup can be investigated to make it automated and more efficient as scope of improvement [7].

ACKNOWLEDGMENT

The authors would like to thank several of their colleagues at NXP who contributed to this work. Special thanks to Inayat Ali and Nancy Amedeo.

REFERENCES

[1] M. Kaur and S. K. Khare, “Achieving Faster Reset Verification Closure with Intelligent Reset Domain Crossings Detection,” in Proceedings of DVCON US, 2020.
[2] Y. Mirsky, “Comprehensive and Automated Static Tool Based Strategies for the Detection and Resolution of Reset Domain Crossings,” in Proceedings of DVCON US, 2017.
[3] “Faster reset verification closure with intelligent reset domain crossings detection,” https://resources.sw.siemens.com/en-US/white-paper.
[4] “Arm® CoreLink™ GIC-600AE Generic Interrupt Controller Technical Reference Manual,” https://documentation-service.arm.com/static/white-paper
[5] S. Mishra, M. Kumar, K. Mishra, A. Jain, and B. V. Gottumukkala, “Quiescent Formal Checks (QFC) for Detecting Deep Design Bugs – Sooner and Faster,” in Proceedings of DVCON US, 2023.
[6] I. Ahmed, K. Nouh, and A. Abbas, “Multiple Reset Domains Verification Using Assertion Based Verification,” in Proceedings of the IFIP/IEEE International Conference on Very Large-Scale Integration (VLSI-SoC), 2017.
[7] M. Handover, “Reset Your Reset Domain Crossing (RDC) Verification with Machine Learning,” in Proceedings of DVCON Europe, 2022