2021 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE OCTOBER 26-27, 2021

Resetting RDC Expectations

A Systematic Approach to Verifying Complex Configurable Designs

Eamonn Quigley, Arm, Cambridge, United Kingdom (*Eamonn.Quigley@arm.com*)

Jonathan Niven, Arm, Cambridge, United Kingdom (*Jonathan.Niven@arm.com*)

Mark Handover, Siemens, Newbury, United Kingdom (*mark.handover@siemens.com*)

***Abstract*—Requirements for asynchronous reset behavior extend the complexities of Clock Domain Crossings as designs add reset domains to meet power and functional requirements. Design configurability common in IP development and in construction of subsystems built of configurable elements introduces challenges to the verification of Reset Domain Crossings (RDC). This paper outlines an approach to RDC verification that efficiently addresses these challenges.**

***Keywords*—CDC, RDC, Clock, Reset, Crossing, Metastability, RTL**

I. Introduction

The challenges of clock domain crossings are understood. However, reset domain crossings often avoid equal consideration in the design process, yet their result is the same – potential device failure. RDC issues must be addressed as the reset architectures of devices become more complex. While the constraints for such an analysis are often more complex than for CDC the general verification process is similar, and it is reasonable to expect that RDC should be a manageable challenge.

However, the challenge addressed by this paper is found in scalability and configurability. A GPU design family, for example, can consist of configurable arrays of repeated blocks. These IP blocks are not considered RDC clean unless their higher-level instantiations are clean. The interactions of configurations and waivers as the designs are analyzed and results studied must be as malleable as the design itself. If unmanaged, the high degree of configurability and scaling results in a nearly unmanageable analysis.

In this environment, a new approach to analyzing RDC results must be adopted in order to ensure program success.

II. Challenges

A. *Degree of configurability*

With highly configurable designs, many iterations of CDC and RDC analyses are run on multiple configurations. Typically, analysis is started on simple configurations with single IP instantiations and eventually run on more complex configurations with multiple instantiations of IP blocks. For example, consider Figure 1. The design consists of a configurable number of slices, in this case headed by blocks labeled *SLn*. Each slice contains a configurable number of sub-blocks. The number of RDC groups vary with configuration, and the potential for cross-slice interactions vary as well. All combinations should be considered such that all possible interactions across CDC groups and RDC groups are verified. Understanding that a CDC or RDC group as seen in Figure 1 consists of many control or data signals, the number of actual crossings that must be verified is substantial and will vary based on configuration.

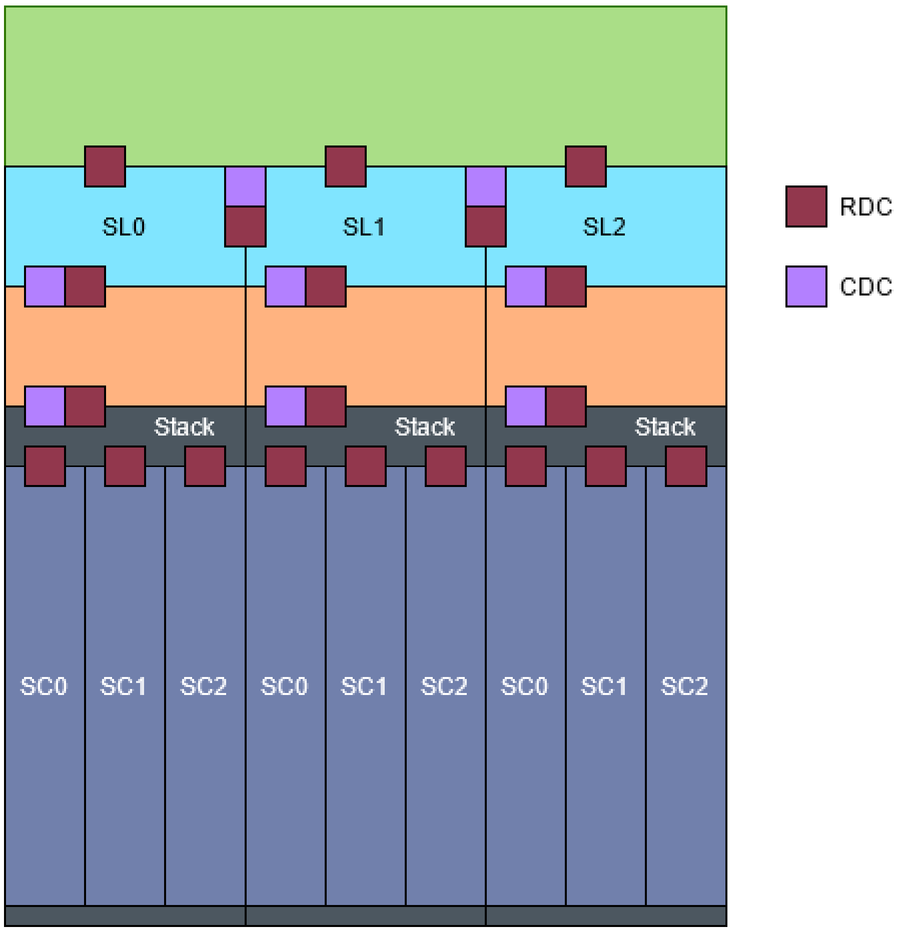

Figure 1 Example of Hierarchical Configurability of Design

In addition to managing the design configurations, it is a challenge to manage the large number of CDC and RDC paths occurring in any large, complex design. The project team must review and address every CDC/RDC violation paths where some paths are common between configurations and some paths are unique to configurations. Efficiently managing violations enables new violations to be detected early, reduces the engineering effort for fixing and verifying paths when detected early and avoids late-stage bug fixes for engineers when they are focused on must-fix tape out issues.

B. Waiver portability

With highly configurable designs, many iterations of CDC and RDC analysis are run on multiple configurations. Typically, analysis is started at the IP level, then progresses to simple top-level configurations with single IP instantiations and eventually progresses to complex top-level configurations with multiple IP instantiations. The waivers generated at the IP level are reused on the simple configurations and the complex configurations. In addition, waivers generated for simple configurations can be extended to apply to complex configurations.

When porting waivers from the IP level to simple and complex top-level configurations, designers must be careful to avoid leniently specifying waivers that may inadvertently hide real design bugs. For this reason, most design teams avoid the lenient behavior of using wildcards in waivers. In some situations, designers could specify a superset of more precise waivers where any configuration may use a subset of the waivers, but this leads to CDC/RDC analysis warnings where waivers are not applied. These warnings can then hide real constraint and waiver errors. The porting of waivers across design hierarchies also creates the challenge of maintaining the connection between a waiver and its correlated bug-tracking ticket.

C. Some RDCs Are Also CDCs

There are efficiencies in parallelizing RDC and CDC verification, but there is duplicated effort when RDC and CDC paths overlap. In the case where RDC and CDC paths overlap, designers will often review the same violation path multiple times.

These overlapping violations include combinational logic violations as seen in Figure 2 where a synchronizer is instantiated on an RDC/CDC path, with the synchronizer impacting reliability when driven by combinational logic. Another example which can occur (but is not present in our designs which are strictly clocked on the positive edge of the clock) is seen in Figure 3 where an RDC/CDC synchronizer with clocks on different phases has reduced reliability due to the reduced metastability recovery time.

III. SOLUTION: A RECOMMENDED CONFIGURABLE APPROACH

Improving RDC efficiency includes taking advantage of synergies between CDC and RDC, leveraging status constraints to manage CDC and RDC results, and generating scalable constraints and waivers.

A. Utilize synergies between CDC and RDC verification

Some RDC and CDC solutions such as Siemens' Questa RDC and Questa CDC leverage a common setup environment. Due to the commonality, it is possible to save time and resources by generating a single setup environment and reviewing common violations once rather than multiple times with separate CDC and RDC runs.

A clean CDC/RDC setup is fundamental to generating accurate and low noise results as well as leveraging the overlap between CDC and RDC verification as identified in Figure 2 and Figure 3. CDC constraints are required for RDC analysis to generate correct clock groups and allow both CDC and RDC to have a common view of the clock and reset trees.

Questa RDC analysis has the capability to identify scenarios where both RDC and CDC violations occur on a CDC path. We leverage this capability in our CDC/RDC project flow by deferring the review of these paths to CDC verification and thus, we avoid the review of the same path during RDC verification. This arrangement to defer CDC/RDC path review to CDC verification has reduced effort by avoiding redundant verification of overlapping paths.

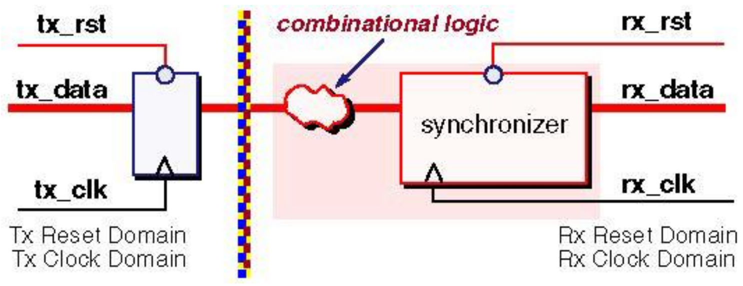

Figure 2 RDC/CDC combinational logic violation

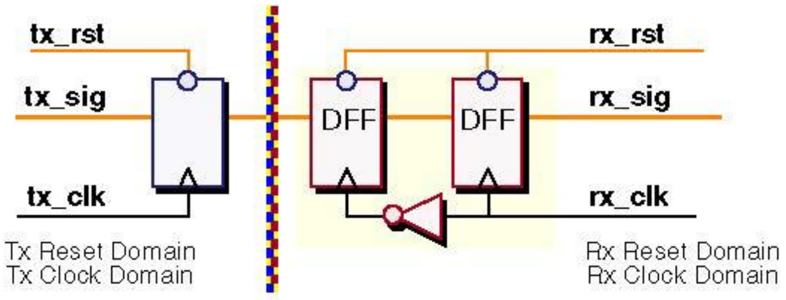

Figure 3 RDC/CDC synchronizer on opposite clock polarity

B. *Managing Results Using Status Flow*

Our project utilized a status constraints management infrastructure to both manage and triage RDC and CDC results. The configurable nature of our design results in both RDC and CDC producing a large number of results, so productivity hinges on efficiently managing these results. Our RDC/CDC status constraints flow does not only indicate waived paths, but the flow provides multiple status states for every CDC and RDC path. We created a flow to utilize the status states to track issues from start through completion. All violations start with a default "uninitialized/uninspected" state. We then change the state after review and debug:

- "Uninspected" state indicates the violation review has not started
- "Bug" state indicates that a ticket has been filed for a violation
- "Pending" state indicates a temporary state for grouping related violations and proposing waivers
- "Fixed" state indicates that an RTL fix is pending
- "Waived" state indicates that a reported violation has been reviewed and is acceptable

Waivers, when created, made use of an embedded messaging scheme to build information into the waiver to accelerate down-stream reviews. The waiver messaging scheme as seen in Figure 4 allowed the grouping of similar violations using a format that tracks the CDC and RDC path information. As seen in the figure, after the waiver number, additional messages are added such as a ticket ID for tracking bug discussions. The embedded messaging scheme makes it easy to explain waivers during design reviews, where the design team presented a few slides on each "type" of waiver, using the "subtypes" as further review collateral. The design team then assessed the explanation and approved the waiver.

Python scripts were written to post-process a comma-separated values (CSV) report and manage the overall flow and closure process. There are 2 primary scripts that drove the Continuous Integration (CI) testing and filtered the crossings specific to bug tickets.

The CI script example seen in Figure 5 read the CSV file and raised an error if any uninspected violations were found.

The filter crossing script example seen in Figure 6 filtered out the CDC or RDC paths related to a particular bug ticket. This technically could be used to filter any arbitrary message, not just bug tickets. The script consolidates the outstanding unresolved crossings and provided an easy way to provide progress updates.

This compartmentalization of results and post-processing flow allows CDC and RDC verification to be incorporated into our continuous integration flow despite the configurability challenges and allows a systematic tracking of issues to enable verification closure.

RDC path message specified as “RDCW-XX-YY-ZZ-AA JIRA-123”:

- XX: Reset scheme: A coded representation of the CDC or RDC structure type (e.g. 0=rdc_areset)
- YY: Type: Primary grouping method used for violations, allows for a level of broader categorization. All violations of the same group usually share a common explanation/overall cause.
- ZZ: Subtype: Allows for a level of granularity within the overall category. For example, paths that share an endpoint would be good candidates for sharing a subgroup.
- AA: A loop iterator. May be considered separate to the previous three, and allows for violations to be separated per iteration. For example, the slice ID would often be recorded, as a lot of violations are per slice.
- JIRA-123: Optional ticket ID for traceability

Figure 4 RDC embedded messaging scheme example

```
 crossings = pandas.read_csv(args.reset_crossings)

    uninspected_crossings = crossings[
        (crossings['Severity'] == 'Violation') &
        (crossings['Status'] == 'Uninspected')
    ]
    num_crossings = len(uninspected_crossings)
```

Figure 5 Python CI script example

```
    crossings = pandas.read_csv(args.reset_crossings)

    filtered_crossings = crossings[
        (crossings['Severity'] == 'Violation') &
        (crossings['Status'] == args.status.title()) &
        (crossings['Comments'].str.contains(args.ticket))
    ]
    columns = ['CheckType', 'Comments', 'TxReset', 'RxReset', 'TxClock',
'RxClock', 'TxSignal', 'RxSignal']
    return filtered_crossings[columns].to_csv(args.outfile, index=False)
```

Figure 6 Python filter crossing script example

C. Generating scalable constraints and waivers

With highly configurable designs, it is a challenge to specify constraints and waivers to IP and top-level instances and signals that apply to multiple instantiations and configurations and yet not result in unintentional application. A scalable constraints and waiver strategy includes converting specifications into constraints, generating constraints to create a clean CDC/RDC setup, and when constraints cannot be used, specifying waivers at the IP-level and top-level.

Ideally, IP-level waivers can be specified at the module level, so the scope for the waiver application will be relative to the module and will be applied at every instantiation of the module. In order to maintain waiver accuracy and avoid unintended waived paths, we avoided the use of wildcards high up in the hierarchical pathnames. For our design situation, we also avoided module-based waivers due to the presence of inter-slice domain crossings between adjacent slices, so the TX register slice or other instance indices and the RX clock domain indices were explicitly specified. The use of conditional Tcl loops accurately and efficiently applied waivers for the analysis of each configuration and avoided both overly aggressive waiver application as well as tool warnings for unneeded waivers.

These conditional Tcl loops as seen in Figure 7 allowed the waivers to both scale to the different configuration runs and be accurately applied to each configuration. The conditional loops expanded the waivers per configuration, and also expanded the embedded messaging comments for each waiver.

```
foreach sl $slices {
        cdc report item -status waived \
            -scheme async_reset_no_sync \
            -message "CDCW-02-01-01-${sl}" \
            -tx_clock clk_top \
            -rx_clock clk_cg_sl${sl} \
            -from
g_gpu_array.u_gpu_array.gpu_slice_${sl}.u_gpu_slice_${sl}.u_global.u_log
ic.u_cgpdc.u_pdc_cg.reset_n_r \
            -module vithar_toplevel

        cdc report item -status waived\
            -scheme async_reset_no_sync \
            -message "CDCW-02-02-01-${sl}" \
            -from
u_always_active.u_dcls_wrapper.u_partition_control.u_partition_manager.g
en_slices\[${sl}\].gen_slice_instance.u_slice_pdc.reset_n_r \
            -to u_always_active.* \
            -tx_clock clk_top \
            -rx_clock clk_cg_sl${sl} \
            -module vithar_toplevel
    }
```

Figure 7 Conditional Tcl waiver loop example

IV. Results

The proposed approach was implemented on a highly complex design with the characteristics shown in Table I.

During our top-level RDC runs consisting of various configurations as seen in Table II, we found that upwards of 5000 RDC paths overlapped with CDC paths. Since our design team thoroughly reviews all CDC paths, these overlapping CDC/RDC paths could be ignored during the RDC analysis. Questa RDC's identification of overlapping paths during RDC analysis improved our design team productivity by avoiding the duplicated review of these paths.

Table I. Design configurability

Level	**Configuration**	**Async Clock Domains**	**Async Reset Domains**
Top-level	1-8 slices	3-17	12-63
Slice	1-3 cores	3	4

Table II. CDC and RDC paths in two sample top-level configurations

Design	**Configuration**	**Total CDC Paths**	**Total RDC Paths**	**CDC/RDC path overlap**
Top-level A	1 core	7137	498313	91
Top-level B	24 cores	629628	13317354	~5000

Table III. CDC and RDC waivers in two sample top-level configurations

Design	**Configuration**	**Total CDC Waivers**	**Total CDC Conditional Waivers**	**Total RDC Waivers**	**Total RDC Conditional Waivers**
Top-level A	1 core	94	71	1857	2458
Top-level B	24 cores	607	584	32320	31643

Our CI flow incorporated the previously mentioned Python scripts to automate the tracking of the large number of CDC and RDC paths. This automation was able to notify the project team of new CDC bugs earlier in the project flow than manual analysis methods and enabled our design team to fix bugs earlier in the project with lower R&D project cost. The CI flow also allowed our project team to better track the CDC and RDC analysis progress and more quickly achieve CDC/RDC verification closure.

The implementation of an embedded messaging scheme integrated within our CI flow enabled our team to manage and address the large number of CDC and RDC waivers in Table III with lower designer effort. The waiver scaling through the conditional Tcl loops reduced the waiver effort and also improved the maintainability of the large number of waivers. The following benefits were realized from the embedded messaging scheme and the waivers in conditional Tcl loops:

- Reduced waiver generation effort by automatically scaling waivers to multi-configuration analysis runs
- Improved CDC/RDC issue tracking and review accuracy by correlating waivers with bug tickets
- Improved design review productivity for reviewing and approving waivers

V. Summary

RDC analysis is made more complex by configurability and scaling, so a different approach that is focused on efficient closure in such an environment is necessary. For complex designs, a Continuous Integration flow is required to verify and manage the large number of CDC/RDC paths and violations. By first eliminating all sources of issues but true RDCs, the team focuses specifically on reset issues. By taking advantage of advanced tool features such as Questa CDC and RDC's status flows and ability to identify CDC/RDC path overlap, the team can triage issues quickly and reduce redundant effort. The use of the embedded message techniques allowed both waiver grouping and correlation as well as bug ticket correlation that in resulted in CI efficiencies and improved design review productivity. Finally, structuring and applying waivers in loops allowed for straightforward adaptation to the configurability in the design.

References

[1] Sulabh Kumar Khare and Ashish Hari, Mentor Graphics, An Automated Systematic CDC Verification Methodology based on SDC setup, DVCon India 2014.
[2] Chris Kwok, Priya Viswanathan, Ping Yeung, "Addressing the Challenges of Reset Verification in SoC Designs", DVCon US, 2015.
[3] Yossi Mirsky, "Comprehensive and Automated Static Tool Based Strategies for the Detection and Resolution of Reset Domain Crossings", DVCon US, 2016.