Avoiding Confounding Configurations

An RDC Methodology for Configurable Designs

Eamonn Quigley, Jonathan Niven, Kurt Takara

Arm Limited, Cambridge UK, Siemens Digital Industries Software, Fremont CA

2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES
VIRTUAL
FEBRUARY 28 - MARCH 3, 2022

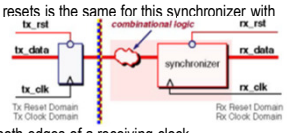
Challenges

1. Asynchronous data or signal crossings are subject to corruption

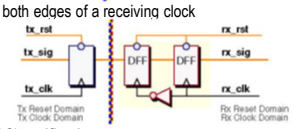
- These are known as Clock Domain Crossings (CDC) or Reset Domain Crossings (RDC). The challenges of CDCs are understood but those of RDCs are often overlooked.
- The corruption is due to a property of a sampling device, known as metastability, that may result in intermediate (non-binary) values for a non-deterministic time after the asynchronous event.
- Safe design practices are known and tools such as CDC or RDC verification verify the presence, and often the correct behavior of synchronizers implementing these best practices. These can not be safely verified in simulation.

2. Some RDCs are also CDCs

- Duplicate review and debug effort occurs when CDC and RDC paths overlap
  - Parallel CDC/RDC runs accelerate compute efforts at the cost of human effort
- For example, the asynchronous domain boundary for clocks and resets is the same for this synchronizer with combinational logic – the CDC is also an RDC

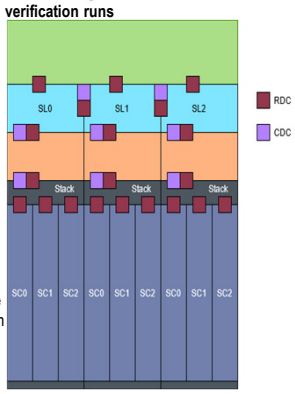


- Another example (but not in our design) is a synchronizer using both edges of a receiving clock



3. Highly configurable designs can require many RDC (and CDC) verification runs

- Configurable number of sub-blocks per slice, slices per IP
- Configuration includes sync/async boundaries
- Number of CDC and RDC groups vary with configuration
  - Potential for cross-slice interactions
- All combinations should be considered across CDC and RDC groups
- Number of crossings to verify is variable but substantial
  - Crossings consist of more than one signal
- Reviewing violations is complex
  - Some violations are common between configurations
  - Others are unique



4. Constraints are not sufficient in a highly configurable design

- Constraints define fixed properties, assumptions, operational space
- Waivers post-process results where a reported issue isn't a problem
- For highly configurable designs, constraints can not completely describe the operational space to preclude all waivers

4. Waivers must be preserved through hierarchy and configurations

- RDC (and CDC) are typically run low-level first, then simple high-level, then complex-configuration high-level
- Issues are missed when flexibility is added with overly-lenient wildcard-based waivers
- Un-applied waiver/constraint warnings occur when a superset of all precise waivers are created and used

Results

During our top-level RDC runs consisting of various configurations as seen in below, 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.

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

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

Our CI flow incorporated the above-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 the table below 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

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

Solutions

1. Eliminate redundant verification by utilizing commonality between CDC and RDC verification

- Use common configuration and status processing found in some tools such as Questa CDC and RDC
  - A clean CDC/RDC setup is fundamental for accurate and low-noise results
  - Verify CDC design and constraints first - CDC constraints are required for RDC analysis
  - Then using clean CDC design and constraints, focus on RDC
- Leverage Questa's identification of joint CDC/RDC paths to defer to CDC, thus focusing on pure RDC

2. Enable Continuous Integration and manage result lifecycle and commonality using tool status flows

- Utilize the Questa constraint and status flows to track issues from start to completion
  - Configurability generates a large amount of issues that we must manage efficiently
- Embed messaging scheme below into waivers to accelerate reviews by grouping similar paths and tracking

Uninspected: review has not started

Bug: a ticket has been filed

Pending: temporary state for grouping and proposed solution

Fixed: RTL fix is pending

Waived: Violation reviewed and accepted

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

- Post-process results to guide Continuous Integration testing and focus on unprocessed new issues by filtering out already-filed crossings

Python CI script example

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

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

Python filter-crossing 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)
```

3. Implement a precise set of waivers and constraints automatically regardless of configuration

- Avoid the use of wildcards to cover instantiation hierarchy
- Avoid the use of module-based waivers due to configuration-based inter-slice domain crossings
- Generated precise waivers for any configuration based on Tcl loops
  - Also expanded waivers to use the embedded messaging comments for each waiver

Conditional Tcl waiver loop example

```
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_logic.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.gen_slices\[${sl}\].gen_sli
ce_instance.u_slice_pdc.reset_n_r \
      -to u_always_active.* \
      -tx_clock clk_top \
      -rx_clock clk_cg_sl${sl} \
      -module vithar_toplevel
 }
```

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.