SIEMENS
Ingenuity for life

Siemens Digital Industries Software

Don’t forget the protocol! A CDC protocol methodology to avoid bugs in silicon

Executive summary

Designers add synchronization logic to prevent the propagation of metastable events, but often designers do not implement or verify the correct CDC protocol. Without a correctly implemented protocol, a CDC structure will not function correctly and lose or corrupt data or propagate metastability. In this paper, we discuss the difficulties encountered with traditional CDC protocol verification methodologies and present an innovative, complete methodology to overcome the current challenges.

Abdelouahab Ayari, Sukriti Bisht, Sulabh Kumar Khare,
Ashish Hari and Kurt Takara

Introduction

Today's complex designs include multiple asynchronous clocks and the signals crossing between asynchronous clock domains may result in functional errors. When a signal from one asynchronous clock domain is sampled by a register on a different asynchronous clock domain, the setup/hold timing requirement will be violated for the destination register. The setup/hold timing violation indicates that the destination register may become metastable and that the destination register will settle to an unpredictable value and possibly cause a functional error. Although clock domain crossing (CDC) verification is a critical task in design verification projects, many design teams only statically verify CDC synchronization structures.

When designers add synchronization logic to prevent the propagation of metastable events, designers should implement and verify the correct CDC protocol. Without a correctly implemented protocol, a CDC structure will not function correctly and thus, lose or corrupt data or propagate metastability. It is common practice for CDC tools to generate assertions to check correct protocol adherence, but assertion generation is not sufficient for designers to verify CDC protocols.

In this paper, we will discuss the difficulties encountered with traditional CDC protocol verification methodologies and present a complete methodology to overcome the current challenges. Following are the key challenges:

- Significant effort and time is required to setup the design and assertions for formal and simulation
- Lack of integration between structural analysis, formal verification and simulation steps
- Considerable effort required to review and debug firings in formal and simulation

The proposed methodology improves ease of setup by automating the setup, constraints and results analysis from static CDC analysis to Formal to Simulation. The automation provided by this methodology avoids manual scripting efforts and thus reduces setup effort and setup errors. The methodology also improves debug productivity by addressing the important issue of correlating structural analysis, formal verification and simulation results.

What are CDC protocols and why are they needed?

Although CDC synchronization structures are required to prevent metastability on CDC paths, designer may overlook the fact that a good structure alone is not sufficient to avoid CDC errors. If a CDC synchronizer is not used correctly, the CDC path may experience data loss or data corruption or in the worst case, metastability. The rules that define the correct usage of a CDC synchronizer are called CDC protocols.

In the simplest protocol case, a 2DFF synchronizer requires a stability protocol where the TX data must be held stable for at least 2 RX clock cycles in order for the RX register to reliably capture the TX data and prevent data loss or data corruption. In a more complex protocol case, a data-mux (DMUX) synchronizer requires that the TX data must be stable when the mux enable is active and allowing the RX register to sample the TX data. If the DMUX synchronizer protocol is violated, the RX register will become metastable even though a correct DMUX structure was implemented.

CDC protocol errors must be identified and addressed early in the design cycle, otherwise they can lead to functional failures in later stages. These functional failures can further result in increased iterations and even silicon re-spins. To ensure such issues are not missed, designers and verification engineers verify synchronizer protocols by generating assertions for synchronizer protocols and verifying them using formal model checking and simulation techniques.

CDC protocol verification challenges

Protocol verification is critical for avoiding CDC errors, but often, project teams do not utilize protocol verification due to the multitude of deployment challenges. The common challenges for deployment include protocol verification setup, constraints setup and the difficulty in reviewing and debugging protocol results.

Protocol assertion generation and verification setup

CDC structural analysis identifies the CDC synchronization structures, then CDC utilities will automatically generate protocol assertions for the CDC synchronization structures. The CDC utilities generate assertion instantiations that connect the TX register, RX register, TX clock, RX clock, TX reset and RX reset to the appropriate protocol assertion.

After CDC protocol assertions are generated, it can be difficult for designers to incorporate these assertions into formal verification and simulation environments. The compilation, formal analysis and simulation commands or scripts are required for running simulation and formal analysis. The clock frequency information from SDC and CDC constraints are used to specify the stability protocol requirement for CDC synchronizer structures.

SystemVerilog assertion and SystemVerilog bind files must be added to the formal and simulation environments. For designers unfamiliar with formal analysis, it is challenging to generate formal compilation and analysis run scripts. For complex simulation environments, it is challenging to incorporate the protocol assertions into simulation regressions.

Formal constraints generation

The formal environment requires designers to specify formal setup constraints that include design configuration information, clock information and input port information. The formal design configuration information includes constants specified for configuration ports and registers. The formal clock information includes the specification of the design clocks and the clock frequencies. The formal port information includes the specification of the primary input ports and their related clock domain information.

CDC protocol assertion debug

In traditional CDC protocol methods, the protocol verification review and debug is isolated from the CDC structural analysis. It is difficult for designers to correlate protocol assertion violations in the simulation or formal environment to the associated CDC synchronization structure. This lack of correlation between structural and protocol verification causes more time and effort spent by designers for review and debug of protocol verification results.

Proposed CDC protocol methodology

In this paper, we propose a methodology that not only helps overcome the common challenges of traditional protocol verification methods, but also better leverages formal model checking and simulation technologies to improve protocol verification results. The proposed methodology has the following workflow, as illustrated in figure 1.

1. Static CDC analysis: Perform static CDC analysis on the design to ensure that all the relevant CDC paths are synchronized using proper synchronizers. In addition, automatically generate protocol assertions, generate formal verification constraints, generate formal verification and simulation setups:

- Automatic protocol assertion generation: Generate assertions for protocols of each synchronizer in the design. Based on static CDC analysis, the CDC protocol generation utility generates protocol assertions for each CDC synchronizer. The protocol assertions are generated depending upon the type of synchronizer and its connections (figure 2)

```
assert_cdc_sync #(2, 1,1) two_dff_81824 (
                .tx_data(demo_top.pass_en[0]),
                .tx_clock(demo_top.cpu_clk_in),
                .rx_clock(demo_top.mac_clk_in),
                .tx_reset(1'b0),
                .rx_reset(1'b0),
                .areset(w_2));
```

Figure 2: CDC protocol assertion instantiation.

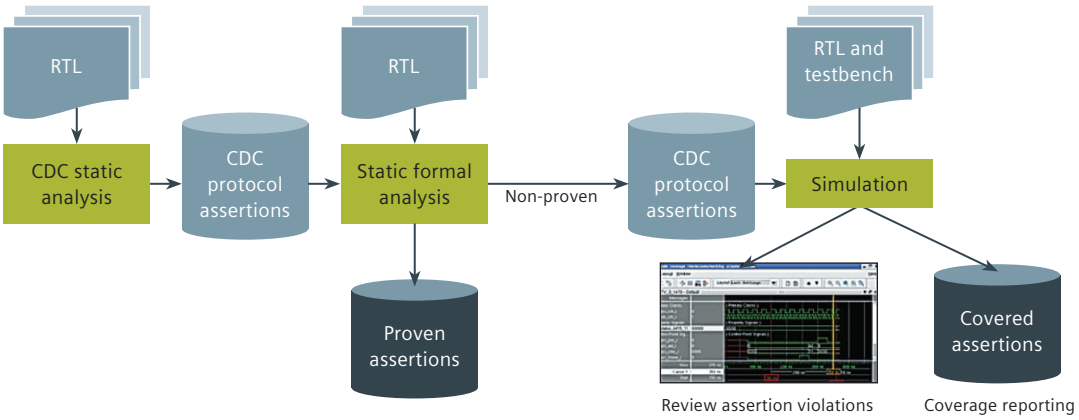

Figure 1: Protocol verification flow.

- Automatic formal verification constraints generation: Generate formal model checking constraints from CDC analysis. The CDC protocol generation utility converts the CDC information for constant, stable, gray-code signals into SVA assumptions for formal verification. In addition, input and output port clock domain information will also be converted into formal constraints to improve the accuracy of formal counter-examples
- Automatic formal verification and simulation setup: Generate setup and scripts for running formal verification and simulation. The CDC protocol generation utility will generate formal model checking run scripts. The utility will also generate a simulation arguments file to be added to designer's existing simulation environment, so designers can easily add protocol assertions and can avoid manually adding multiple assertion and bind module files to their simulation.

2. Formal analysis: Run formal model checking to verify the auto-generated synchronizer protocol assertions using the generated formal verification setup and script. The CDC protocol generation utility generates the formal compile and run scripts. The automated formal setup significantly reduces the effort required to setup the design for formal analysis and also avoids the debug effort to resolve incomplete or incorrect setup issues. In order to run formal analysis, designers simply execute the makefile (i.e. "make all").
3. Simulation: Run simulation by adding the auto-generated setup to the existing simulation environment to verify the non-proven protocol assertions. The CDC protocol generation utility generates simulation arguments files for compilation and simulation.
   During the formal analysis, the formal analysis script automatically updates the simulation setup to remove the proven assertions from the list of assertions for simulation. If formal analysis is run without formal design constraints, a formal proof for any protocol assertion indicates that the protocol can never be violated for the associated CDC synchronization structure. This proof indicates that the synchronization structure is safe from data loss, data corruption and the propagation of metastability. Removing the proven assertions from simulation will reduce the simulation runtime and reduce the designer effort for reviewing simulation results for proven assertions that cannot be violated in simulation.
4. Review and debug CDC protocol results: Review the aggregated structural CDC analysis, formal verification and simulation results. The correlation between the multiple technologies (static CDC analysis, formal model checking and simulation) enables designers to more quickly and easily debug and resolve protocol errors. This correlation enables designers to find the CDC structure associated with a protocol failure, then quickly diagnose the cause of the protocol error. This improved review and debug ensures that bugs are correctly analyzed and protocol errors for CDC synchronizers are not missed or incorrectly analyzed.

Real world results

This CDC protocol verification methodology was applied on a few real world designs. The deployment of the methodology on real life designs provided evidence that this methodology significantly reduced design and verification engineers' effort and helped to achieve faster design closure.

Using a traditional protocol verification methodology, the protocol assertions were validated first in a formal verification environment. The design settings and constraints like clocks, resets, constants and input port constraints were manually translated from the static CDC environment and re-specified as formal constraints. This constraints translation required significant manual effort and formal expertise. Once the formal tool environment setup was complete, the design was verified formally and the assertion firing counter-examples were debugged.

Similarly, in the proposed methodology, the first step was protocol assertion verification in formal verification environment. The design configurations and constraints for formal analysis were auto-generated by using the proposed methodology. This auto-generated setup was reviewed and used to run formal analysis on the design and assertions. The setup automation avoided the need for manual setup effort and formal expertise that would be required by a traditional approach to port the setup from CDC to formal.

Formal verification results included proofs, firings and inconclusives for CDC protocol assertions. The auto-generated formal setup did not include design constraints, so the firings produced unconstrained formal assertion violations that are in most cases, counter-examples with illegal stimulus sets. For designers without formal model checking expertise, it is difficult to debug and/or resolve fired and inconclusive assertions, so instead of debugging these difficult cases, designers will promote these assertions to simulation. With the traditional methodologies, proven protocol assertions are re-verified in simulation environment or must be manually removed from the simulation setup.

In the proposed methodology, only non-proven assertions from formal analysis are verified in simulation. The proven assertions are automatically removed from the simulation setup. The auto-generated simulation arguments file is added to the existing simulation environment. Any simulation assertion firings indicate a violation of the CDC synchronizer protocol that would result in data loss, data corruption or metastability propagation on the associated CDC path. Designers must debug simulation firings and fix the CDC logic to adhere to the synchronizer protocol rules. Significant reduction in debug effort and time was observed due to running simulation only for formal non-proven assertions.

Tables 1 and 2 summarize the comparison between the traditional and proposed methodology performed on a set of real life designs, ranging from 1 to 30 million gates.

The following improvements resulted:

- Significant reduction in formal setup time. The setup time was reduced due to the automated setup generation. In addition to the reduction in setup time, the setup debug effort was reduced by avoiding the incremental debug iterations of incomplete and incorrect setups. The setup time was reduced by 3x-5x.
- Reduction in setup time for simulation. The setup time was reduced due to the automated setup generation. In addition to the reduction in setup time, the setup debug effort was reduced by avoiding the incremental debug iterations of incomplete and incorrect setups. For design C, the setup time was reduced by 6x-17x.

Table 1. Results using traditional protocol verification methodology

CDC protocol verification with formal						
Design	Setup time	Assertions			Run time	Formal coverage*
		Total	Failed	Proven		
A	1w 3d	170	79	91	24s	100%
B	2w 2d	800	304	437	5hrs	92.6%
C	5w 4d	8552	5673	877	7hrs	76.6%

CDC protocol verification with simulation					
Design	Setup time	Assertions		Run time	Simulation tool coverage**
		Total	Failed		
A	10min	170	14	20min	91.7%
B	17min	800	83	35min	88.3%
C	30min	8552	127	1hr	79.4%

* Formal coverage = ((Failed assertions + proven assertions) / Total assertions) * 100
* Simulation coverage = ((Failed assertions + covered assertions) / Total assertions) * 100

Table 2. Results using proposed protocol verification methodology

CDC protocol verification with formal						
Design	Setup time	Assertions			Run time	Formal coverage*
		Total	Failed	Proven		
A	2d	170	32	138	55s	100%
B	5d	800	119	654	6hrs	96.6%
C	1w	8552	1825	4907	10hrs	78.7%

CDC protocol verification with simulation					
Design	Setup time	Assertions		Run time	Simulation tool coverage**
		Total	Failed		
A	1min	32	6	20min	97.31%
B	1min	146	34	35min	97.06%
C	5min	3645	76	1hr	87.47%

* Formal coverage = ((Failed assertions + proven assertions) / Total assertions) * 100
* Simulation coverage = ((Failed assertions + covered assertions + proven assertions) / Total assertions) * 100

- Reduction in false firings in formal analysis. The automated setup generation and the import of CDC design constraints into formal analysis reduced the formal setup errors and caused a reduction in false firings. The formal assertion firings were reduced by 59-68 percent.
- Increased formal coverage. The improved formal setup and constraints resulted in less inconclusive assertions and more proofs and firings.
- Increased simulation coverage. Removing proven assertions from simulation resulted in a higher percentage of fired and covered simulation assertions. Since the proven assertion were not simulated in the proposed methodology, the proven assertions were considered covered in order to maintain simulation coverage consistency between the traditional and proposed methodologies.
- Reduction in simulation verification time and effort. There was a reduction in the number of assertions passed to simulation due to exclusion of formally proven assertions thereby reducing the verification effort required for reviewing proven assertions in simulation. The correlation of structural CDC analysis, formal verification and simulation results improved debug productivity and led to easier association of protocol errors with their associated CDC paths. The time and effort for reviewing and debugging simulation results was not measured for these design case studies.
- Reduction in effort for debugging false simulation firings. In the proposed methodology, there was a reduction in the number of simulation assertion failures. In the proposed methodology, the CDC constraints used in formal analysis allowed more assertion proofs for cases where stable or gray-coding constraints enabled assertion proofs. These proven constraints were not run in simulation, so the designers would avoid debugging false simulation assertion firings. The time and effort for debugging false simulation firings was not measured for these design cases.

From the real life designs, there was a minimal change in simulation runtime. The reduction in the number of assertions run in simulation due to exclusion of formally proven assertions did not significantly change the simulation runtime.

Conclusion

The proposed CDC protocol verification methodology described in this paper is a systematic and repeatable solution for the design and verification of CDC synchronizer protocols. The automated setup process reduces the setup errors and the debug iterations required to resolve setup issues. Automated conversion of CDC constraints into formal model checking constraints improves the accuracy of formal analysis and reduces false formal firings. Finally, the integrated CDC structural analysis, formal verification and simulation results enables an easier to use debug environment that allows designers to debug and fix protocol errors more quickly and with less effort.

References

1. Mark Litterick, "Pragmatic simulation-based verification of clock domain crossing signals and jitter using System Verilog Assertions," Verilab.
2. William K. Lam, "Hardware design verification: simulation and formal method-based approaches," Prentice Hall, Mar 3, 2005.
3. Sulabh K. Khare, Ashish Hari, "Systematic speedup techniques for functional CDC verification closure", Siemens Digital Industries Software, advanced verification white paper.
4. Ping Yeung, "Five steps to quality CDC verification," Siemens Digital Industries Software, advanced verification white paper.
5. Sukriti Bisht, Sulabh K. Khare, Ashish Hari, "A systematic take on addressing dynamic CDC verification challenges", DVCon US, 2019.

Siemens Digital Industries Software

Headquarters
Granite Park One
5800 Granite Parkway
Suite 600
Plano, TX 75024
USA
+1 972 987 3000

Americas
Granite Park One
5800 Granite Parkway
Suite 600
Plano, TX 75024
USA
+1 314 264 8499

Europe
Stephenson House
Sir William Siemens Square
Frimley, Camberley
Surrey, GU16 8QD
+44 (0) 1276 413200

Asia-Pacific
Unit 901-902, 9/F
Tower B, Manulife Financial Centre
223-231 Wai Yip Street, Kwun Tong
Kowloon, Hong Kong
+852 2230 3333

About Siemens Digital Industries Software

Siemens Digital Industries Software is driving transformation to enable a digital enterprise where engineering, manufacturing and electronics design meet tomorrow. The Xcelerator portfolio helps companies of all sizes create and leverage digital twins that provide organizations with new insights, opportunities and levels of automation to drive innovation. For more information on Siemens Digital Industries Software products and services, visit www.sw.siemens.com or follow us on LinkedIn, Twitter, Facebook and Instagram. Siemens Digital Industries Software – Where today meets tomorrow.

siemens.com/eda

81998-C3 4/20 N TGB