

# Pragmatic Formal Verification Methodology for Clock Domain Crossing (CDC)

Aman Kumar, Infineon Technologies, Dresden, Germany (*aman.kumar@infineon.com*)

Muhammad Ul Haque Khan, Cadence Design Systems, Munich, Germany (*muhammad@cadence.com*)

Bijitendra Mitra, Cadence Design Systems, Bangalore, India (*bijitm@cadence.com*)

**Abstract** —Modern System-on-Chip (SoC) designs are becoming more and more complex due to the technology upscaling. SoC designs often operate on multiple asynchronous clock domains, further adding to the complexity of the overall design. To make the devices power efficient, designers take a Globally-Asynchronous Locally-Synchronous (GALS) approach that creates multiple asynchronous domains. These Clock Domain Crossings (CDC) are prone to metastability effects, and functional verification of such CDC is very important to ensure that no bug escapes. Conventional verification methods, such as register transfer level (RTL) simulations and static timing analysis, are not enough to address these CDC issues, which may lead to verification gaps. Additionally, identifying these CDC-related bugs is very time-consuming and is one of the most common reasons for costly silicon re-spins [1]. This paper is focused on the development of a pragmatic formal verification methodology to minimize the CDC issues by exercising Metastability Injection (MSI) in different CDC paths.

**Keywords** —*formal verification; clock domain crossing; metastability*

## I. INTRODUCTION

Integrated Circuits (ICs) are a fundamental component that is essential for modern electronic devices. They can perform a wide range of functions, from signal processing to artificial intelligence, and are used in many industries. However, as ICs have become more complex, verification of their designs has become a major challenge [1]. In fact, design verification now takes up a significant amount of project time and failures to identify all functional issues can result in delays and even setbacks in terms of expensive silicon re-spin. A recent study conducted by Siemens and Wilson Research Group shows that design verification consumes approximately 60% of the total project time [1]. Given the fierce competition in the semiconductor market and ongoing problems like chip shortages, a single silicon re-spin could cost a lot of money and prolong the product's time to market by months, significantly decreasing the chip's market share and profit potential. In the study [1] from Fig. 1, flaws in clocking are the third largest contributor towards the re-spin. One particular challenge is CDC, which occurs when signals cross asynchronous clock domains in intricate IC designs. To ensure that ICs are tolerant of these problems and function properly, new verification methods need to be devised.

Furthermore, the GALS approach increases the overall CDC complexity of the design. The state-of-the-art verification techniques such as conventional RTL simulation or Static Timing Analysis (STA) alone are not enough to address the CDC issues of such intricate SoC designs. Hence, it is essential to devise a new verification methodology to address the CDC issues of such complex designs. The new CDC verification methodology should ensure that the Design Under Verification (DUV) is tolerant to problems such as metastability effects, re-convergence errors, Reset Domain Crossing (RDC) issues, and data loss faults. This basically implies that in the design, the CDC signals can span across asynchronous clock domains without being missed or wrongly sampled, thereby preventing metastable values from propagating further downstream.

## II. TECHNICAL BACKGROUND

A Clock Domain (CD) is a part of the design that is driven by one or more clocks that are synchronous to each other (i.e., active edges are aligned). However, most real-life designs operate on multiple asynchronous clocks. A CD boundary arises when the clocks change. There are several reasons for using multiple CDs [2]:

- Power and performance folding: High-frequency logic often requires more power than lower frequency logic. Therefore, designs often have a larger amount of lower-speed logic than faster-clocked logic. The design is more energy efficient while operating parts of a circuit with a lower frequency clock than the other part of the

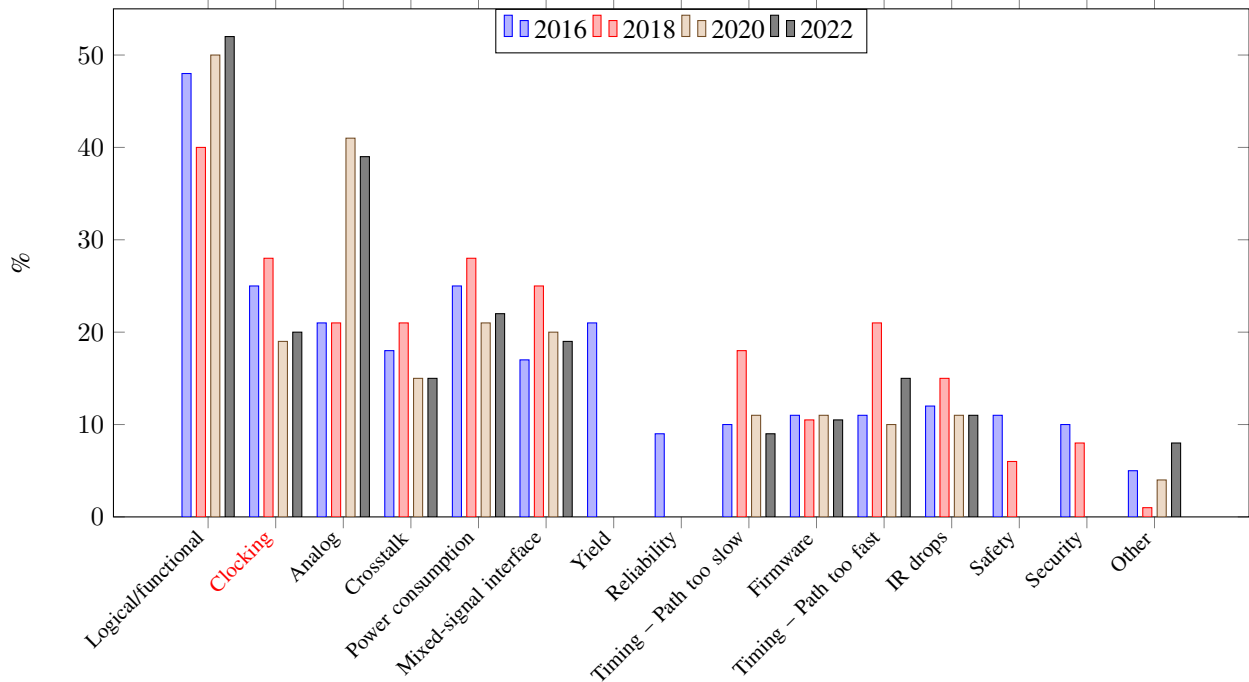


Fig. 1: Type of ASIC flaws contributing to re-spin [1]

design. In a low-power design, one can turn off one CD while keeping other CDs active, thus reducing power consumption.

- Physically separate clocks: Usually each system has its own clock generator to make sure that they work unambiguously even if networking cables, such as Ethernet, are disconnected. Quartz-based crystal oscillators generally used as clocks will differ in accuracy with temperature, supply voltage, and crystal age. However, this creates different clock domains at the receiver end.

There are several issues associated with designs that have multiple CDCs. These problems are well-documented in [3] including [4]:

- Setup and hold time violations due to metastability
- Functional errors due to convergence of synchronized signals
- Functional errors due to divergence through multiple synchronizers

#### A. Metastability

Metastability is a state of instability in a circuit where a flip-flop output cannot settle to a stable '0' or '1' logic level within the required time to function correctly. This can result in intermediate voltage levels being processed incorrectly and the circuit remaining in an unstable condition for a period of time, resulting in functional failures.

#### B. Causes of Metastability

Metastable behaviour occurs in flip-flops due to setup and hold timing violations. The setup time is the minimum amount of time during which the input data must arrive and be stable before the active edge of the clock. When the input data changes during the setup time window, a setup violation occurs. The hold time is the minimum amount of time during which the input data must be stable after the active edge of the clock. When the input data changes during the hold time window, a hold violation is observed. When a setup or hold timing violation occurs, the output of the flip-flop may go metastable as shown in Fig. 2. For CDC signals coming from asynchronous clock domains, it is not possible to prevent metastability from occurring due to the non-deterministic clock relationships.

Setup/hold violations cannot usually be detected in an RTL simulation. When there is a setup violation as depicted in Fig. 2, the simulation always captures a '1', while in silicon it may produce a '0' or '1'. As a result, during setup violation, the output transition of the flip-flop may get delayed by a cycle in silicon. When there is a hold violation

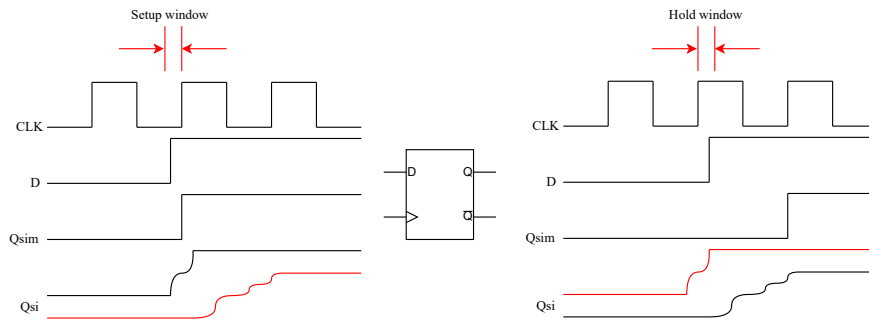


Fig. 2: Setup and hold time violation [5]

as depicted in the above figure, the simulation always captures a ‘0’, whereas in silicon it may produce either ‘0’ or ‘1’. During hold violation, the output transition of the flip-flop may occur one cycle early in the silicon. A flip-flop becomes unstable when setup or hold conditions are violated, and it eventually settles down to either ‘0’ or ‘1’ logic level, after an unpredictable delay.

### C. Effects of Metastability

The metastability effect can have the following consequences:

- It can lead to unanticipated system behaviour or functional failures.
- Propagation of metastable data (unstable data) to different blocks of the SoC design can induce high currents and can eventually result in chip burnout.
- The design may enter into an unknown state or into a deadlock scenario due to the metastable behaviour of the control signals.

In multi-clock chip architectures, metastability cannot be avoided, but its detrimental effects can be mitigated by using the right methods, such as the use of synchronizers for clock domain crossing signals [6]. To minimize issues related to metastability propagation and data loss, different analysis such as structural CDC analysis, verification of the CDC protocol and functional CDC analysis are performed [4].

## III. CONVENTIONAL CDC VERIFICATION METHODOLOGY

In the conventional CDC verification process, as depicted in Fig. 3, design and verification are treated separately. Once the design specification is complete, the design engineer creates the RTL implementation of the code, which is then checked for structural clock domain crossover issues using a static CDC tool. The analysis looks for problems such as missing or incorrectly placed synchronizers and glitches caused by combinatorial logic on the synchronizer paths. To facilitate this analysis, the designer writes a series of CDC constraints that may include specifying CDC false paths, identifying static and constant signals, and so on. Once the structural analysis is completed, the designers can optionally perform a functional analysis to ensure that the synchronizer protocols are correct. This is the standard CDC sign-off process used by designers.

The work of a functional verification engineer starts with the creation of a test plan that aligns with the design specification. Subsequently, the engineer creates a simulation testbench and develops test cases to validate the functionality of the design. If all the test cases of the regression suite are passing, the verification engineer may choose to run the simulation using randomizable synchronizer models, although this scenario is not typical during the pre-silicon functional verification stage due to the limited time budget. Finally, the verification engineer declares the CDC verification sign-off when both code and functional coverage have reached 100%. However, there are a few drawbacks to the conventional verification approach. The gaps associated with the CDC sign-off flow of designers are:

- When the design engineer writes CDC constraints manually, it is error-prone, and a wrong CDC constraint or assumption could lead to an incorrect structural analysis by the static CDC analysis tool. This may let a functional bug slip into the post-silicon phase, resulting in expensive silicon re-spins.
- There is no functional analysis for user-defined synchronizer schemes.
- Absence of functional analysis with metastability modelling required to detect issues such as metastability propagation and re-convergence errors.

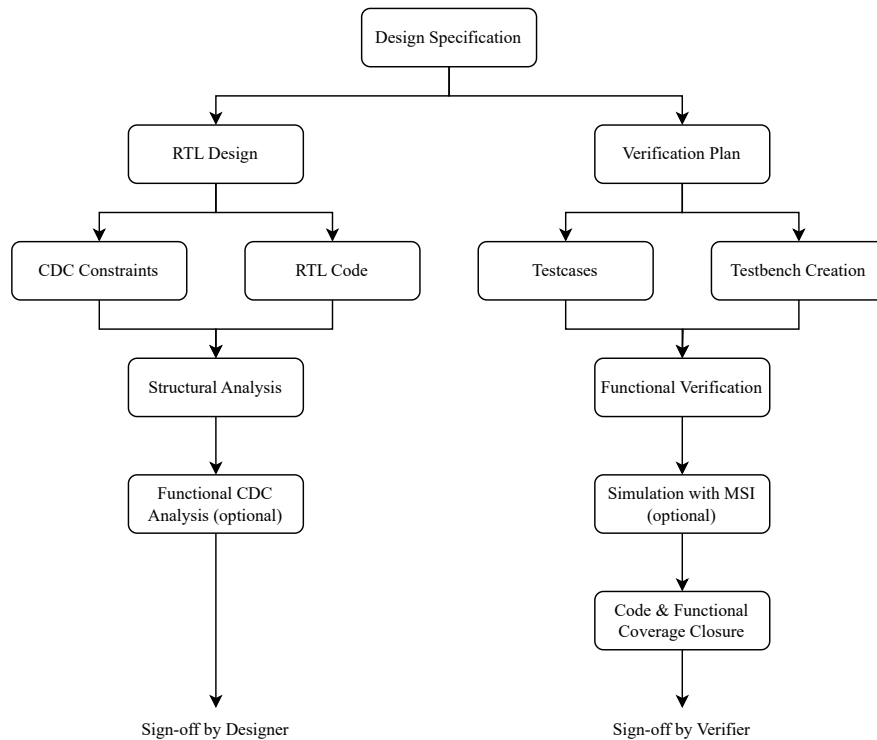


Fig. 3: Conventional CDC verification methodology

There are also gaps associated with the CDC sign-off flow of verification engineers. The verification engineer usually does not perform formal functional verification with MSI, which is a significant reason why the functional bug escapes to the silicon. The second major gap is the use of inaccurate MSI models in simulation runs. Most standard MSI models only mimic setup violations, which are incomplete and insufficient to detect all CDC issues. Third, no coverage model has been defined to measure the completeness of the CDC verification, which is crucial to determine the sign-off quality of the DUV. The proposed verification methodology in this paper uses a combination of both formal (static) and simulation (dynamic) based verification methods to address the shortcomings of conventional CDC verification methodologies.

#### IV. PROPOSED CDC VERIFICATION METHODOLOGY

The drawbacks of conventional CDC verification methodology clearly highlight that there is a missing link between CDC sign-off and functional verification sign-off flows that usually run as independent activities, creating verification gaps. Here, the verification gaps are created. Our proposed flow addresses these gaps efficiently and effectively.

To overcome the drawbacks, we propose a more pragmatic verification methodology based on formal approach as shown in Fig. 4. Formal verification has advantages over simulation based verification as highlighted in [7], [8] and [9]. Although, there are associated limitations as well as mentioned in [7] and [9] but with the right approach, formal verification could bring in a lot of advantages.

We use a metamodel-based automation framework [11] to prove the functional correctness of the design at the IP level, use MSI models to prove the correctness under metastability influence, generate CDC coverage model and later on, use the same MSI and properties setup in simulation-based verification to prove the testcases under metastability influence at SoC level.

##### A. Sign-off flow for CDC Design

The CDC design process begins with the implementation of the design unit in RTL based on the architecture specification, using clock-oriented partitioning to mitigate CDC problems. Next, a structural CDC analysis is performed on the RTL code, where basic CDC issues such as missing synchronizers, incorrectly placed synchronizers, glitches induced by combinatorial logic in the synchronizer paths and convergence or divergence errors are identified.

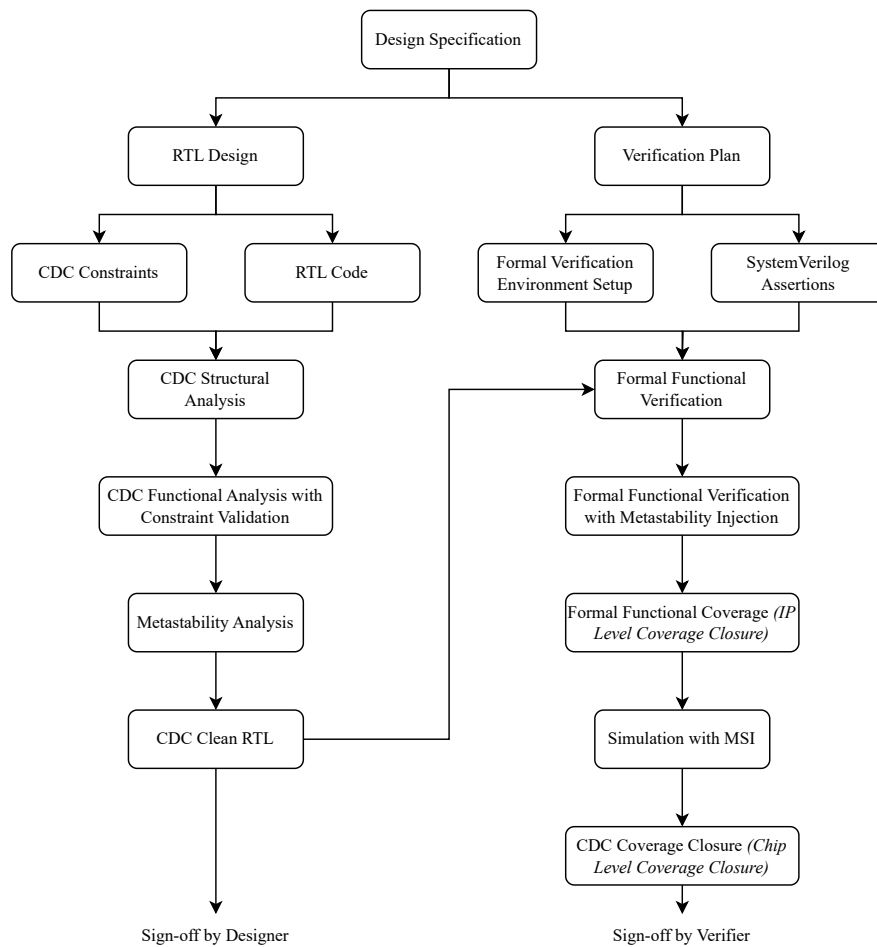


Fig. 4: Proposed CDC verification methodology [10]

It is also important for the designer to provide the CDC constraints that define legal/illegal behaviour premises for structural CDC analysis. Afterwards, a functional analysis is performed to ensure proper synchronization protocols and prevent CDC issues such as data loss. Often designers may black-box parts of the DUV to simplify the proof process, however, an incorrect black-box may hide crucial CDC related bugs. SystemVerilog assertions generated by the CDC code generator tool are used to verify the CDC constraints, ensuring no incorrect black-boxing of the design. Finally, a formal functional analysis with metastability injection is performed using a comprehensive MSI model to verify the synchronization scheme without metastability-induced errors. This sign-off flow for the CDC design should be followed to produce the CDC-compliant RTL code, and an example of a CDC constraint check (SystemVerilog assertion) generated by the CDC code generator tool is provided in Listing 1.

---

```

1 // static signal check with condition
2 property static_check_sig;
3     (!$stable(sig)
4     |->
5     (!sig_static_cond));
6 endproperty
7 assert property(@posedge clk) static_check_sig);

```

---

Listing 1: Example code for CDC constraint check

### B. Sign-off flow for CDC Verification

The verification sign-off flow employs a formal verification setup to verify the design at the IP level, followed by simulation-based testing at the SoC level. Fig. 4 depicts the required steps for the CDC verification closure of multi-clock asynchronous designs. The flow includes both formal-based methods to verify the design at the Intellectual Property (IP) level and at the same time, reuse of IP level setup at the SoC level to perform CDC verification with MSI.

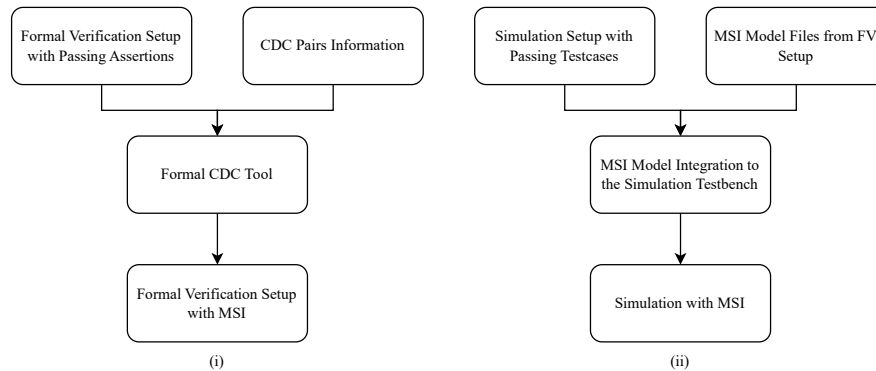


Fig. 5: Metastability injection flow in (i) formal verification setup and (ii) simulation setup

#### 1) Formal Verification Setup

The CDC verification flow begins with creating a plan based on the design specification, including functional features to address. SystemVerilog assertions are written by the verification engineer to verify these features and are complemented by assertions from the CDC code generator based on synchronizer schemes. The formal verification setup checks these assertions using CDC-compliant RTL code. The next step involves validating the assertions with metastability injection to ensure proper functionality despite metastability effects. Finally, the CDC coverage model generated by the CDC generator tool is analyzed to sign off the verification at the IP level. Fig. 5 (i) shows the metastability injection flow in the formal verification setup. To enable MSI support, the formal CDC tool, in our case Cadence Jasper needs two inputs: the formal setup with passing assertions and information about CDC pairs in the design. The tool collects information on CDC pairs during structural analysis and does not need the user to explicitly provide this information. The MSI model generated by the tool automatically handles both the setup and hold timing violations for every CDC pair. Since we start with passing assertions, any assertion failure after running MSI enabled formal proofs clearly indicates metastability effects to be the reason for failure.

#### 2) Simulation Setup

The SoC level design is verified using a simulation-based setup. The process starts with a simulation regression to verify functional features at the chip level. Once all regression test cases pass, a metastability injected regression ensures the design works correctly in presence of metastability effect. Additionally, CDC functional checks (SystemVerilog assertions) from the formal verification setup are exported to the simulation to validate the synchronizer protocols in the presence of metastability effects. The CDC coverage model is then used to assess the completeness of CDC verification at the SoC level. Fig. 5 (ii) illustrates the metastability injection flow in the simulation setup. The inputs to this flow are a simulation setup with passing test cases and MSI model files exported from the formal verification setup (Cadence Jasper). Integrating the MSI model involves instantiating it in the testbench top file. This represents the process of metastability injection in simulation, where we used Cadence Xcelium as the tool of choice.

### C. CDC Coverage Model

A CDC coverage model is developed to analyze the completeness of CDC verification as illustrated in Fig. 6

The coverage model consists of two levels of coverage:

- IP level coverage - indicates the CDC pairs coverage at the block level and is evaluated using a formal verification setup.

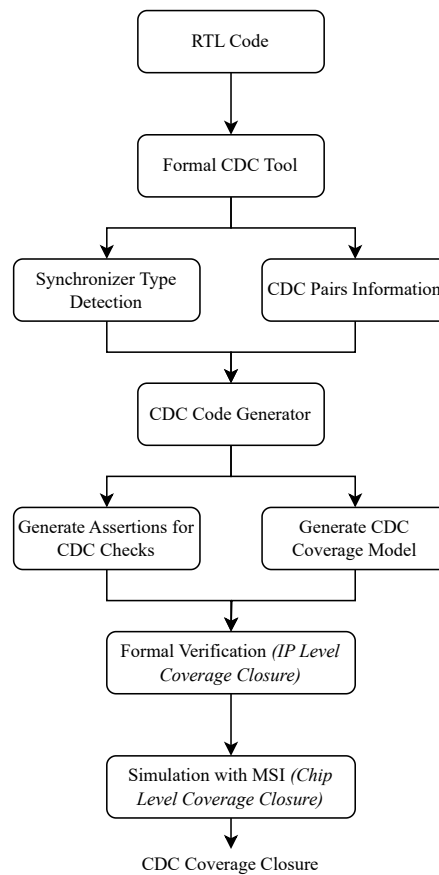


Fig. 6: CDC coverage model

- SoC level coverage - indicates the CDC pairs coverage at the chip level and is evaluated using a simulation-based setup with MSI.

The CDC code generator tool requires two inputs: synchronizer types and CDC pairs information from the CDC-compliant RTL code. The tool generates the CDC coverage model (Listing 2) for formal verification. The developed coverage model analyzes the coverage of the following cases for all CDC pairs:

- Setup time violation resulting in a logic 0 at the destination flop
- Setup time violation resulting in a logic 1 at the destination flop
- Hold time violation resulting in a logic 0 at the destination flop
- Hold time violation resulting in a logic 1 at the destination flop

#### D. CDC Code Generator

The CDC code generator tool is developed to generate both the CDC checks (SystemVerilog assertions) and the CDC coverage model. As shown in Fig. 7, the CDC code generator requires two inputs to generate the CDC checks and the coverage model. The first input is the details of the synchronizers present in the design. The Jasper formal CDC app is used to extract this information about synchronizers from the design. The second input is the CDC pair information, which is also extracted similarly from the Jasper CDC app.

The CDC code generator tool produces SystemVerilog assertions for various CDC checks, including functional analysis of synchronizer schemes, clock gating checks, and signal configuration checks. It supports standard synchronizer schemes (2-DFF, data path, pulse type, mux-based, multi-bit) and some custom synchronizers. The tool also generates signal configuration checks to validate CDC constraints set during structural CDC analysis. Additionally, it creates a CDC coverage model to analyze the coverage of CDC pairs in the design. An example of the SystemVerilog assertions generated by the tool is shown in the Listing 3.

---

```

1 // CDC coverage model
2 module cdc_coverage;
3 // CDC pair signals extracted from the CDC generator tool
4 logic sig_src_1;
5 logic sig_dest_1;
6 // coverpoints are sampled wrt clock 'clk'
7 covergroup cg @(posedge clk);
8 // coverage for CDC pair sig_src_1 and sig_dest_1
9 cp_src_1 : coverpoint sig_src_1;
10 cp_dest_1 : coverpoint sig_dest_1;
11 cx_cdc_pair_1 : cross cp_src_1, cp_dest_1;
12 endgroup
13 // instance for CDC coverage
14 cg cdc_cover_inst = new();
15 endmodule
  
```

---

Listing 2: Generated CDC coverage model

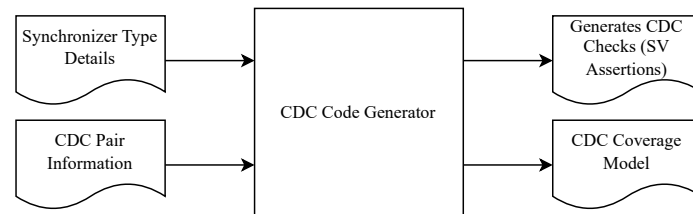


Fig. 7: CDC code generator

---

```

1 // pulse synchronizer checks
2 // assumptions/constraints
3 property pulse_input_width_pulse_i;
4 @(posedge prim_clk_i)
5   disable iff (!prim_reset_n_i)
6   $rose(pulse_i)
7   |->
8   ##1 !(pulse_i)[*2] ##1 (pulse_i) ##1 !(pulse_i) ##1 !(pulse_i);
9 endproperty
10 assume property(pulse_input_width_pulse_i);
11
12 // 1. output pulse width check
13 property pulse_output_width_check_pulse_o;
14 @(posedge sec_clk_i) disable iff (!sec_reset_n_i)
15   pulse_o | => !pulse_o;
16 endproperty
17 assert property(pulse_output_width_check_pulse_o);
18
19 // 2. pulse transfer check
20 property pulse_transfer_check_pulse_i;
21 @(posedge prim_clk_i) disable iff (!prim_reset_n_i || !sec_reset_n_i)
22   pulse_i |-> @(posedge sec_clk_i) ##[0:3] oulse_o ##1 !pulse_o;
23 endproperty
24 assert property(pulse_transfer_check_pulse_i);
  
```

---

Listing 3: Generated assertions by CDC code generator



The code generator tool is developed using the metamodel-based automation framework [11]. Fig. 8 displays the UML representation of the metamodel, illustrating the relationships between its classes. The generator's metamodel includes separate classes for each synchronizer type and a class responsible for generating the coverage model.

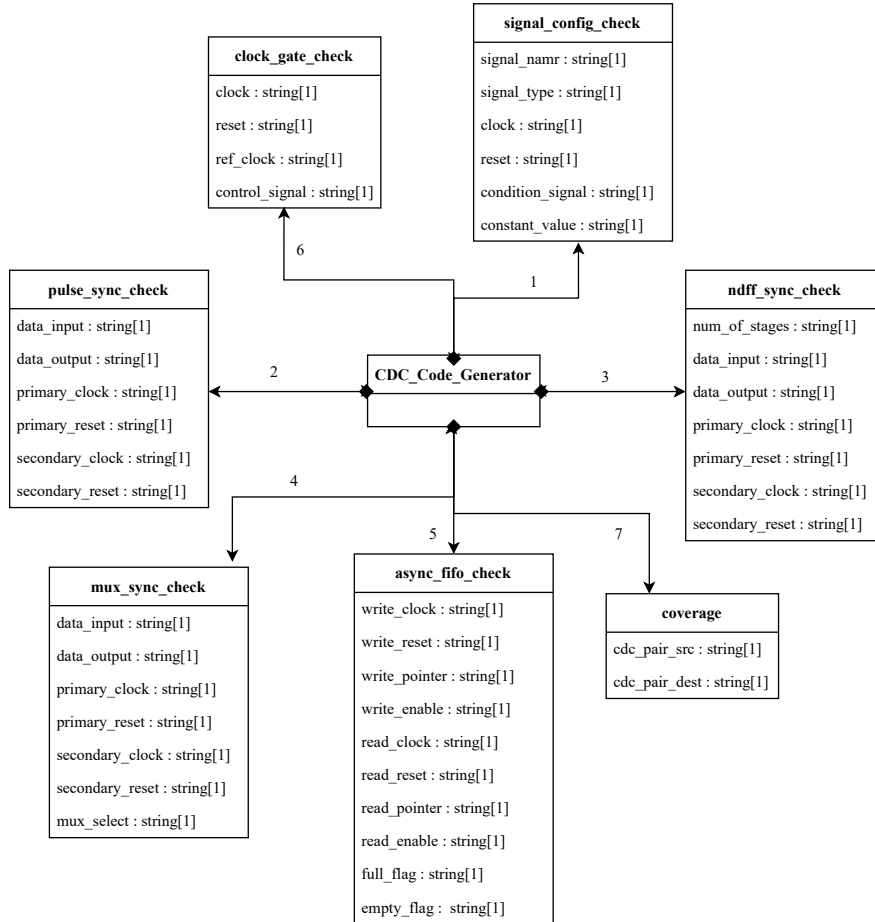


Fig. 8: UML class diagram of the CDC code generator

The metamodel of the generator consists of the following models with distinct functionalities:

- signal\_config\_check - generates checks for signal configuration and CDC constraints specified by the designer.
- pulse\_sync\_check - generates checks for pulse synchronizers
- ndff\_sync\_check - generates checks for NDNF synchronizers, where 'N' denotes the number of flip-flops used
- mux\_sync\_check - generates checks for multiplexer-based synchronizers
- async\_fifo\_check - generates checks for asynchronous FIFO
- clock\_gate\_check - generates checks for clock gating feature
- coverage - generates CDC coverage model

After generating the metamodel, the framework parses it to create Python-based Application Programming Interfaces (APIs) and Graphical User Interface (GUI) files of the tool. The GUI of the 'CDC code generator' is then invoked to set the values of the class attributes and finally, the specification is saved in an XML format. The CDC checks and coverage model are finally generated in two separate files afterwards.

## V. RESULTS

The methodology successfully verified the design units for CDC issues at the IP and chip levels. Formal CDC verification with metastability injection, along with CDC coverage analysis, was done successfully for the IP level CDC verification. Similarly, simulation with metastability injection and CDC coverage analysis was performed successfully for the chip-level CDC verification.

The application of the verification methodology developed on different design units such as common cells, ethernet controller IP, and an SPI block working in different clock domains unveiled several hard-to-find bugs. It was successful in finding many RTL design bugs and testbench bugs. The types of bugs identified with this verification methodology are shown in Table I.

Bug Type	Analysis Used	Bug Description
RTL bug	Structural analysis	Missing synchronizer for CDC signal
RTL bug	Structural analysis	Missing synchronizer for RDC signal
RTL bug	Structural analysis	Combinational logic on the CDC path
RTL bug	Structural analysis	Combinational logic on the RDC path
RTL bug	Structural analysis	Reset signal converged before reaching the destination unit
RTL bug	Functional analysis	Wrong signal configuration (signal wasn't static)
RTL bug	Functional analysis	Signal not stable enough to be captured correctly by destination clock
Testbench bug	Functional analysis	Input pulse was more than 1 cycle wide (pulse synchronizer)
Testbench bug	Functional analysis	Data loss because of incorrect clock frequencies
Testbench bug	Metastability analysis	Assertion didn't handle the extra delay due to metastability propagation
Formal tool bug	MSI model generation	MSI model was not getting generated for the simulation

TABLE I: Types of CDC bugs detected

In addition to finding the above-mentioned bugs, the methodology also analyzed the CDC pairs coverage using the developed CDC coverage model and the one generated by Cadence Jasper [5]. As shown in Fig. 9, the methodology was successful in finding many CDC pairs with extremely poor coverage. This shows that the developed methodology is efficient in finding potential CDC issues.

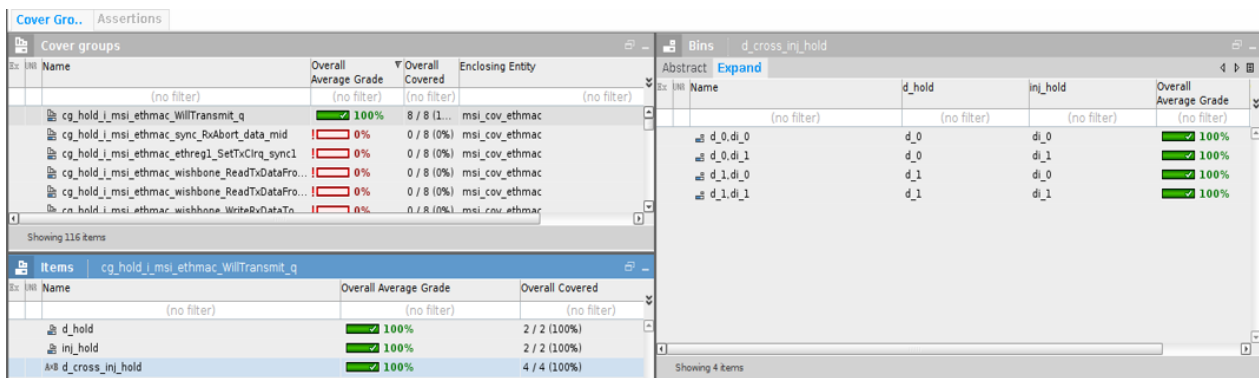


Fig. 9: CDC pairs coverage

## VI. CONCLUSION

In this paper, we presented a verification methodology for multi-clock SoC architectures to address CDC issues. The methodology bridges the gap between structural and functional verification and employs simulation and formal verification methods, as well as metastability aware verification, to detect all CDC design bugs. The methodology consists of five stages of CDC analysis: structural, functional, metastability, simulation with metastability injection, and CDC coverage analysis. The methodology successfully detected hard-to-find corner case CDC-related issues and reduced the designer's effort in generating the jitter model by offering a comprehensive metastability injection model. In general, the methodology is exhaustive, efficient, measurable, and scalable.

## REFERENCES

- [1] Harry Foster. *2020 Wilson Research Group Functional Verification Study*. Tech. rep. Mentor, A Siemens Business, Oct. 2020.
- [2] David J. Greaves. *Modern System-on-Chip Design on Arm*. Arm Education Media, 2021.
- [3] R. Ginosar. "Fourteen ways to fool your synchronizer". In: *Ninth International Symposium on Asynchronous Circuits and Systems, 2003. Proceedings*. 2003, pp. 89–96. DOI: 10.1109/ASYNC.2003.1199169.
- [4] Mark Litterick. "Pragmatic Simulation-Based Verification of Clock Domain Crossing Signals and Jitter using SystemVerilog Assertions". In: *DVCon US, 2006*.
- [5] *Jasper CDC User Guide*. Tech. rep. Cadence, Jan. 2023.

- [6] Clifford E. Cummings. “Clock Domain Crossing (CDC) Design & Verification Techniques Using SystemVerilog”. In: SNUG, 2008.
- [7] Erik Seligman, Tom Schubert, and M V Achutha Kiran Kumar. *Formal Verification, An Essential Toolkit for Modern VLSI Design*. Morgan Kaufmann Publishers, 2015.
- [8] Aman Kumar and Sebastian Simon. “A Semi-Formal Verification Methodology for Efficient Configuration Coverage of Highly Configurable Digital Designs”. In: DVCon US, 2021.
- [9] Aman Kumar. “Pragmatic Formal Verification of Sequential Error Detection and Correction Codes (ECCs) used in Safety-Critical Design”. In: DVCon US, 2023.
- [10] Binil Mathews. “Development of a Verification Methodology for Clock Domain Crossing (CDC)”. MA thesis. Germany: Darmstadt University of Applied Sciences, 2022.
- [11] Keerthikumara Devarajegowda. “Model-based Generation of Assertions for Pre-silicon Verification”. doctoralthesis. Technische Universität Kaiserslautern, 2021, pp. V, 167. DOI: 10.26204/KLUEDO/6640. URL: <http://nbn-resolving.de/urn:nbn:de:hbz:386-kluedo-66403>.