Systematically Achieving CDC Verification Closure based on
Coverage Models and Coverage Metrics

Ashish Hari (ashish_hari@mentor.com), Yogesh Badaya (yogesh_badaya@mentor.com)

Mentor Graphics Corporation

ABSTRACT

Clock domain crossing (CDC) verification is a critical step in functional verification closure. Design complexity is continuously increasing, not only in size but also in heterogeneity of system-level design. A typical design includes complex features such as multiple cores, peripheral devices, many types of IO-interfaces, power islands, and so on. Handling such technologies requires the use of multiple asynchronous clock domains. As the number of clock domains increases, the probability of encountering CDC issues increases considerably—thereby making CDC verification a critical development step.

Increasing the time to complete the verification process or having degraded verification quality can significantly increase overall costs. Both factors depend on the approach used for CDC verification. Persistent CDC verification challenges are: tracking progress and identifying when to stop. One puzzling question for verification engineers is: "Is my CDC Verification done?"

In this paper, we propose a novel methodology to overcome verification closure challenges. This methodology gives verification teams reliable information, plus a way to identify measurable goals, to measure progress and to decide when to stop. We define 'coverage metrics' and 'coverage models' for CDC verification closure:

1. *Coverage metric is an indicator of progress at each step of verification flow.*

2. *Coverage models are models defined for the various categories of CDC problems. They ensure that the synchronizers and protocols at clock domain crossings are comprehensively verified for all variations and assumptions. Characteristics of CDC synchronizers and transfer protocols require a directed coverage strategy and directed targets. We also suggest coverage models for ensuring metastability tolerance and analyzing CDC jitter.*

We provide SystemVerilog CDC models for solving common CDC problems that an engineer can use along with their design files in a normal verification flow.

1 INTRODUCTION

Complete CDC Verification is a 5-phase process:

1. **Design Setup Validation**

   Verify the clock trees and the design configuration.

2. **Structural Clock Domain Crossing Analysis**

   Find missing and incorrect synchronizers.

3. **Synchronizer Protocol Verification**

   Use simulation and formal verification to verify the transfer protocols of clock domain crossings, including advanced protocols such as handshake schemes and FIFO synchronization.

4. **Reconvergence Verification**

   Ensure safe reconvergence of synchronized signals by testing the gray encoding of converging signals

5. **Metastability Tolerance Verification**

   Verify that the design tolerates CDC jitter and metastability effects. CDC synchronizers handle metastability correctly, but with a possible side effect of introducing unpredictable delays. Design functionality should tolerate the presence of metastability effects and clock jitter.

To know when verification is complete, you must define verification targets for each phase. The selection of these coverage targets depends on the design's application domain and of course, your intended time-to-market.

We propose a coverage-based CDC verification flow. Here, the verification team sets measurable goals for each verification phase. To do this, we provide specific coverage numbers used to measure validation. These coverage numbers are attained through coverage metrics and coverage models that we have defined for each verification phase. The verification team can use the coverage numbers for the current verification phase to decide whether or not to advance to the next phase. We also propose an overall coverage metric that reflects the current quality of the entire CDC verification process.

2 CDC COVERAGE METRICS AND COVERAGE MODELS

First, we define *coverage metric* for each phase of the CDC verification process. Each phase has different criteria for determining coverage. For example, phases 1 and 2 use static CDC analysis data to identify coverage. Phases 3, 4 and 5 use coverage data collected by coverage models during standard verification. Verification teams can set coverage goals for each CDC verification phase and for the final CDC coverage.

A good coverage metric should model and cover each CDC verification phase, should provide clear information about the current status of the verification process, and should direct the verification focus to valid problems. Such coverage metrics help the verification engineer fix design problems and add new directed tests that cover missing scenarios. A diagram for this coverage model and metric-based CDC verification approach is shown in *Figure 1*.
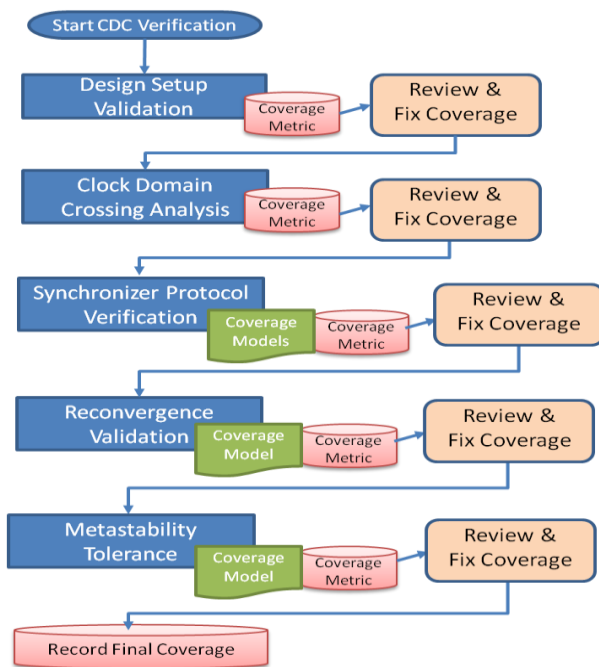

Figure 1: Coverage-based CDC Verification Flow

Coverage criteria, metrics and verification targets for each CDC verification phase are defined in following sections.

2.1 PHASE 1: DESIGN SETUP VALIDATION

The goal of the design setup validation phase is to validate the clock tree and to tune design configurations for CDC analysis. Satisfactory closure of the setup is critical to minimize noise and maximize verification efficiency prior to advancing to static CDC analysis.

2.1.1 Clock Tree Verification

During clock analysis, the CDC verification tool identifies the asynchronous clock trees in the design. For a design to function correctly, it is important to do the clock connectivity checks. All flops should be clocked by user-specified primary clocks. Inferred clocks can be one of the following:

- **Direct Clocks**

  Primary ports or black box output pins that drive clock logic. The verification engineer should verify that these signals are valid clock signals missed as user-specified clocks.

- **Gated Clocks**

  Combinations of enable/clock signals that drive clock logic. The verification engineer should identify clock enable signals and mark them as stable (so no gated clocks are found). Otherwise, intentional gated clocks should be waived as qualified clocks.

- **MUXed Clocks**

  Multiple clocks reaching into the clock logic through a multiplexer that selects one of the clocks. MUXed clocks are selected through mode or configuration settings. The multiplexer *select* signal should be set to appropriate constant constraint settings to ensure the correct clock is enabled.

Verification Target: Verification engineer must resolve (or waive after qualification) all gated, MUXed and inferred clocks to attain coverage closure for this sub-phase.

2.1.2 Black Box Qualification

The following types of design blocks are considered to be *black boxes* by the CDC analysis tool:

- IP blocks.
- Blocks those are designed separately and are not yet ready to be integrated.
- Blocks that are designated by the user to be treated as black boxes and should be skipped by CDC analysis, such as PLL, ADC blocks, and so on.

Verification Target: Verification engineer must review all identified black boxes. For coverage closure, qualified black boxes in the design should be waived—so the percentage of unqualified black boxes should be 0%.

2.1.3 Design Component Classification

Sequential design components should be able to correctly sample their data. Where data sampling is blocked, sequential design components can be classified as follows:

- Stuck at the same value.

  Sequential cell (such as a flop, latch or memory) whose clock is tied to a constant.

- Never changes value.

  A sequential cell with a data pin connected to a constant and the set/reset logic is missing.

The sequential cells in these categories might be okay per the design requirements or user-specified constraints. But, the verification engineer must check and waive them as qualified so that they do not contribute to the uncovered portion of the coverage data.

We define the percentage *overall coverage* for the design setup validation phase as:

$$C1 = \frac{St - (Su + Sc + Sd)}{St} \times 100$$

Where:

St: Total count of sequential cells

Su: Count of unconstrained sequential cells (not clocked by a qualified clock)

Sc: Count of sequential cells driven by constant clock pin

Sd: Count of sequential cells with constant data and no reset condition

See *Figure 2* for a sample classification of design setup coverage data.
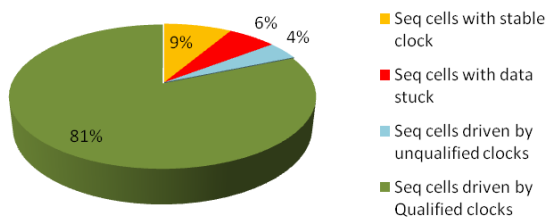
Design Setup Coverage

Figure 2: Sample Clock Tree Coverage Data

Verification Target: Verification engineer must review coverage figures and after excluding sequential cells driven by qualified clocks, coverage should reduce close to 0%. Acceptable coverage at this phase ensures that the clock tree is verified, design configuration is reviewed, and the design has no sequential components that are stuck unintentionally. The design is set up to proceed with phase 2 of CDC verification (clock domain crossing analysis).

2.2 PHASE 2: CLOCK DOMAIN CROSSING ANALYSIS

A *clock domain crossing*, or CDC, occurs when a signal generated in one clock domain is sampled in another clock domain that is asynchronous to the first. In certain cases, such paths can violate the setup/hold time requirements at the receiving registers. These violations can occur in random cycles and a design must be resistant to such effects.

Metastability is the inability of a flop to arrive at a known state in a specific amount of time when the setup or hold conditions are violated. Metastability cannot be avoided—but its effects can be mitigated by use of an appropriate synchronizer at the clock domain crossing.

The goal of the clock domain analysis phase is to ensure that valid synchronizers exist at all CDC paths. Key information extracted for this phase is classified as follows:

- Missing Synchronizers
- Incorrect Synchronizers
- Good Synchronizers

"Missing" and "good" synchronizers are self-explanatory. An "incorrect synchronizer" is one where either the synchronizer circuit is partially correct or the prescribed synchronizer for that type of crossing is not used. For example, a CDC path with 2-flops in the receiving domain is a good synchronizer (*Figure 3a*).
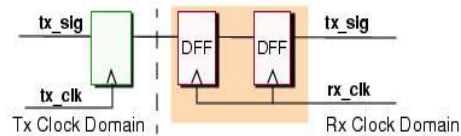

Figure 3a: Good 2-flop Synchronizer for Scalar CDC Signal

However, the presence of combinational logic between the transmit and receive flops makes it an incorrect synchronizer—as it allows glitches to be fed directly to the synchronizing structure—see *Figure 3b*
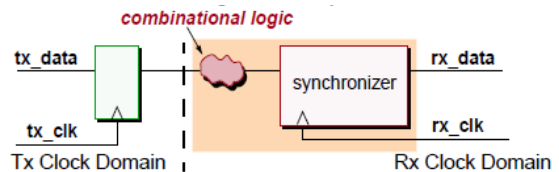

Figure 3b: Incorrect Synchronizer with Combo Logic

Similarly, using a 2-flop synchronizer for individual bits of a bus can lead to data coherency issues at the receiver. The 2-flop synchronizer should be replaced by a MUX-synchronization scheme *(Figure 4)*.
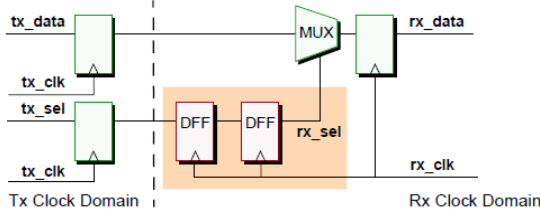
Figure 4: MUX synchronizer is Recommended for Vectors

Coverage for the CDC analysis phase is defined as:

$$C2 = \frac{Xt - (k1Xm + k2Xi)}{Xt} \times 100$$

where:

Xt: Total clock domain crossings

Xm: Crossings with missing synchronizers

Xi: Crossings with incorrect synchronizers

k1, k2: Weights—to be adjusted based on application or design requirements

Coverage for the CDC analysis phase can be improved:

- Typically, designers add valid synchronizers that prevent metastability effects from propagating through the design. So, the verification engineer must review *missing synchronizer* warnings. If a transmitting signal is quasi-static or stable, waive the associated warning.

- Replace incorrect synchronizers with valid ones.

Verification Target: Verification engineer calculates the coverage target for the CDC analysis phase based on the design application and project criticality. Acceptable coverage at this phase ensures all CDC paths have good synchronization structures and the design can properly handle the effects of metastability at clock domain crossings.

2.3 PHASE 3: SYNCHRONIZER PROTOCOL VERIFICATION

Every synchronizer has some properties (or assumptions) that need to be functionally verified. Violating any of these properties might result in data loss at the crossing paths, which can eventually lead to functional failure.

The synchronizer protocol verification phase ensures that the CDC paths' synchronizer protocols are never violated. To accomplish this, synchronizer properties are promoted as assertions for the synchronized CDC paths. A verification engineer can verify synchronization protocol assertions: 1) using simulation with self-checking test benches that match actual with expected results, or 2) through formal methods.

A failure of a protocol assertion (sometimes called a firing) indicates that the design can malfunction. The cause of the firing must be fixed. Once simulation does not cause a CDC protocol firing, the verification engineer must ensure that all CDC paths are sufficiently tested by monitoring coverage. Such testing minimizes the possibility of undiscovered bugs in the design.

Protocol coverage is accomplished through SystemVerilog coverage constructs in the CDC transfer protocol properties. Such data can be gathered using a simulation or formal verification tool that supports these SystemVerilog constructs. We next define SystemVerilog models for a standard set of synchronizers to validate their CDC transfer protocols and collect coverage data.

2.3.1 2-Flop Synchronizer

One protocol for a 2-flop synchronizer (*Figures 3a* and *5a*) checks that the transmit signal is held stable long enough for its value to be captured reliably by the receiver. The protocol also ensures no data loss.

```
property cdc_stable;
    @(posdedge txclk)
        !$stable(tx) |=> $stable(tx)[*2 /* tx_min_cycles*/ ];
endproperty : cdc_stable

assert property (cdc_stable);
```

Figure 5a: SVA for 2-Flop Synchronizer

Here, *tx_min_cycles* is the minimum number of *txclk* cycles during which *tx* must remain stable for its value to be captured reliably by the receiver. This minimum cycle count is calculated from the ratio of the periods of the receiving and transmitting clocks.

2.3.2 MUX Synchronizer

The protocol for the MUX-synchronization scheme (*Figures 4* and *5b*) checks that the signal between two clock domains is held stable long enough for the signal to be sampled reliably by the receiver. The data must remain stable while the data select signal (*enable*) asserts.

Equivalent checks:

- Transmit select signal should not assert for less than 2 clock cycles in the receiving clock domain.

- Data signal should not change while the select (*enable*) signal is asserted.

```
property cdc_stable;
    @(posdedge txclk)
        !$stable(tx) |=> $stable(tx)[*2 /*tx_min_cycles*/ ];
endproperty : cdc_stable

property data_stable_while_enable(data, enable, clk)
    @(posedge clk)
        $rose(enable) |=> $stable(data)[*1:max] ##0 !enable;
endproperty : data_stable_while_enable

assert property (cdc_stable);
assert property (data_stable_while_enable(data,
                                    rx_enable, rx_clk));
```

Figure 5b: SVA for Mux synchronizer

2.3.3 Handshake Synchronizer

Handshake synchronization scheme properties (*Figures 5c and 5d*) verify that the handshake protocol between a transmitter and receiver is correctly obeyed and that the transfer data are stable in the data transfer window.

Equivalent checks:

- Data are stable when *request* is asserted (*data_stable*).
- Every *request* gets an *acknowledge* within the next two cycles (*req_has_ack*).
- No *acknowledge* is issued without a *request* (*ack_had_req*).

```
property data_stable;
    @(posdedge clk)
        req |=>$stable(data) [*1:max]##0 ack;
endproperty  : data_stable

sequence req_ack_seq;
    @(posdedge clk)
        req ##1 !req [*1:max] ##0 ack;
endsequence : req_ack_seq

property req_has_ack;
    @(posedge clk)
        req |->req_ack_seq;
endproperty : req_has_ack

property ack_had_req;
    @(posedge clk)
        ack |->req_ack_seq.ended;
endproperty : ack_had_req

assert property (data_stable);
assert property ( req_has_ack);
assert property ( ack_had_req);
```
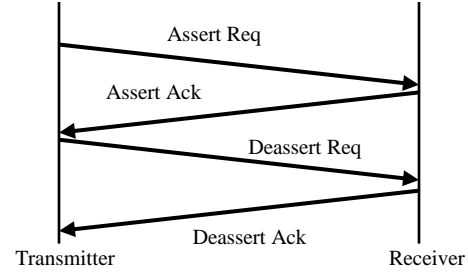
Figure 5c: SVA for Handshake Protocol Checks

Figure 5d: Handshake Scheme

2.3.4 FIFO Synchronizer

The FIFO synchronization protocol ensures that the write and read pointers of an asynchronous FIFO (*Figures 5e and 5f*) change by a hamming distance of 1 and that the FIFO does not overflow or underflow.

Equivalent checks:

- Overflow: no *write* when full
- Underflow: no *read* when empty
- Gray-encoding: *read* and *write* pointers are gray-encoded at the source

Figure 5e: FIFO Scheme

```
property bad_access(clk, inc, flag)
    @(posedge clk)
        inc |-> !flag;
endproperty : bad_access

property gray_code(clk, rst, data)
    @(posedge_clk) disable_iff (rst)
        !$stable(data) |-> $onehot( data ^ $past(data));
endproperty : gray_code

assert property(bad_access (wr_clk, wr_inc, fifo_full);
assert property(bad_access (rd_clk, rd_inc, fifo_empty);
assert property(gray_code(wr_clk, wr_rst, waddr);
assert property(gray_code(rd_clk, rd_rst, raddr);
```

Figure 5f: SVA for FIFO Protocol Checks

Coverage for the synchronizer protocol verification phase has three types of metrics:

- **Protocol Coverage**

  Protocol coverage is defined as:

  $$C3 = \frac{Pt - Pu}{Pt} \times 100$$

  where:

  > *Pt: Total promoted protocols*
  >
  > *Pu: Uncovered Checkers*

- **Synchronizer Coverage**

  Verification team should assign a coverage metric for each type of synchronizer. Reviewing synchronizer coverage identifies some basic issues and dead-code conditions in the synchronizer implementations.

- **Check Coverage**

  Multiple properties can be associated with a synchronizer protocol. For example, the FIFO scheme has *overflow*, *underflow* and *data stability* assertions associated with its CDC transfer protocol. Each protocol property must be covered. For example, missing coverage for a particular protocol property might camouflage a non-functional path in one of its associated synchronizers. So, adding directed tests to validate this special scenario is necessary to attain check coverage closure.

Verification Target: Verification engineer reviews and tracks the protocol coverage metric at all levels. Coverage for protocol verification can be improved by: 1) fixing possible dead-code and error conditions in synchronizer implementation and 2) adding directed test cases targeting uncovered scenarios. Acceptable coverage at the synchronizer protocol verification phase ensures that all synchronized paths obey the protocols and no CDC transfer event can cause a functional error.

2.4 PHASE 4: RECONVERGENCE VERIFICATION

The reconvergence of synchronized signals can lead to data-coherency issues and to subsequent functional errors (if timing dependency exists between the reconverging CDC paths). Since a synchronizer can introduce unpredictable latency on its CDC path, the design logic must be sufficiently resilient to tolerate any resulting data incoherencies. For example, reconverging signals could be gray-encoded to avoid timing dependency between signals.

Verification of reconvergence takes two steps:

1. Run static CDC analysis to identify reconvergence of synchronized signals. Unless required, avoid reconvergence of synchronized signals.

2. Where reconvergence is intentional, check the gray encoding on the reconverging signals. For example, promote gray-encoding protocol checks on the reconverging signals and verify them in simulation.

Coverage for the gray-encoding protocol check at each reconvergence point is collected by standard SystemVerilog cover properties.

Cumulative coverage for the reconvergence verification phase is defined as:

$$C4 = \frac{Rt - Ru}{Rt} \times 100$$

where:

> *Rt: Total reconvergence conditions (excluding waived or structurally fixed cases)*
>
> *Ru: Uncovered checkers for gray-encoding checks*

Verification Target: Verification engineer assigns and verifies a coverage metric that ensures coherency issues at reconverging points are expected and do not lead to functional errors. This reconvergence coverage is improved by 1) removing reconvergence conditions or altering the design logic, or 2) ensuring that diverging synchronized signals are always gray encoded before they reconverge.

2.5 PHASE 5: METASTABILITY TOLERANCE VERIFICATION

Even when a design is structurally verified and simulation runs correctly, metastability in the design's silicon implementation might cause functional errors. So, to accurately model silicon behavior, we use metastability models on the CDC receive registers.

A *metastability model* injects metastability on a bit of a bus for one cycle. Metastability models are used during simulation to model design behavior in the presence of metastability effects on CDC paths. They also have cover properties that identify cases where different forms of metastability have been exercised during simulation. For coverage closure, these assertions must be effectively covered.

Coverage for the metastability tolerance verification phase is defined as:

$$C5 = \frac{Mt - Mu}{Mt} \times 100$$

where:

> *Mt: Total CDC paths for which metastability model is inserted*
>
> *Mu: Uncovered checkers*

Verification Target: Verification engineer must ensure metastability coverage has achieved an acceptable level for which the design is resilient enough to handle random metastability effects occurring in silicon. In our systematic, phased approach to CDC verification, we proceed to the next phase only after achieving the coverage target of the current phase. As it is the last phase, metastability tolerance verification coverage closure guarantees satisfactory overall CDC coverage.

2.6 OVERALL CDC COVERAGE

The sequential nature of our verification methodology means we proceed to subsequent phases in the CDC verification flow only after previous phases are "clean."

For identifying issues and improving the testbench suite, coverage analysis and coverage closure are critical at each phase. However, an *overall CDC coverage* metric can have special significance as a measure of CDC verification quality. This overall coverage value does not provide information for debug, but it can give a fair sense of quality. Such an overall coverage metric should reflect the need for, and the importance of, the sequential CDC verification flow and closure.

Based on experiments and the sequential nature of our CDC verification flow, we define *final CDC coverage* as a weighed mean of coverage metrics across all phases—with higher weights for the initial phases:

$$C = k1C1 + k2C2 + k3C3 + k4C4 + k5C5$$

where:

> *C1 to C5: Coverage values figures for the verification phases*
>
> *K1 to K5: Weights, such that:*
>
> > *k1 ≥ k2 ≥ k3 ≥ k4 ≥ k5*
> >
> > *k1 + k2 + k3 + k4 + k5 = 1*

Weights are set for the particular application and are based on priorities determined by the verification team. For our experiments:

> *k1 = 0.3, k2 = 0.3, k3 = 0.2, k4 = 0.1, k5 = 0.1*

3 COVERAGE MODEL DESIGN

The CDC coverage models for the various checks described in this paper are written in SystemVerilog using constructs such as sequences, properties, assertions and cover statements. Each model is a separate SystemVerilog module. Designers connect models to the actual signals in the design through SystemVerilog *bind* statements. The models are non-intrusive and do not require modification of the golden design. Each model contains the following sections:

- **Protocol Checks**
  - Properties to check assumptions about the correctness of the synchronizer functionality.
  - Coverage data collected for the properties to ensure they are triggered and verified during the verification flow.

- **Coverage Checks**
  - Cover properties to ensure all situations of verification are getting covered on the synchronizer.
  - Cover groups to collect coverage data in a more organized structure and to gather statistics on the synchronizer's verification.

- **Debug Data**

  Statistics collected about synchronizer functionality. Such information is useful when debugging a protocol violation and when inspecting coverage holes.

- **Control Flags**

  Flags that completely or partially enable/disable various features in the models. Verification engineers might want to use control flags for some features to reduce the impact of the coverage models on simulation performance.

4 COVERAGE-BASED CDC VERIFICATION FLOW AND CASE STUDY

For this case study, a coverage metric is defined for each phase of the CDC verification flow. Coverage models are used to collect relevant information as the CDC verification phases are executed. A target coverage for each CDC verification phase is pre-defined and verification proceeds to next phase only if the target of the current phase is achieved. This process results in a systematic and comprehensive coverage-based CDC verification flow.

We took an industry design case-study to illustrate the benefits of this approach. Coverage models were used for validation of the different types of synchronizer protocols and to perform reconvergence and metastability tolerance checks. Coverage data at each phase was analyzed to decide whether or not the CDC verification for that phase was complete. The details of the phased coverage metrics are represented in *Appendix A*.

The overall coverage value appeared lower, as there was insufficient coverage for phase 2 of the CDC verification flow (structural CDC Analysis). And, since this is an early phase in the flow, it had a bigger impact on the overall coverage numbers.

When we set strict coverage goals and proceeded to the next phase only after attaining coverage closure at the current phase, results were much better.

- **Phase 1**

  When reviewing the coverage figures, we identified a module with 232 flops that was incorrectly instantiated—its clock port was connected to a dangling wire. As a result, 232 flops in design were unconstrained. After fixing the issue, the revised coverage at phase1 increased to 100%.

- **Phase 2**

  We added some missing synchronizers and waived off all warnings caused by stable transfer signals. The coverage for phase 2 improved to 98.1%.

- **Phases 3 to 5**

  We performed regular reviews of coverage data and addressed associated problems, which also improved the coverage figures.

Overall coverage improved to 97.1% with regular review of coverage metrics based on the coverage models. Phase coverage information (in conjunction with the overall CDC coverage metric) helped provide a clear answer to the puzzling question— "Is my CDC verification done?"

Rather than relying on a verification engineer's whimsical judgment, we reviewed the coverage data and concluded emphatically: "Yes…CDC verification is DONE!"

5 CONCLUSION

Our proposed methodology helps achieve systematic, accurate and reliable CDC verification closure. Coverage models accurately confirm functional verification of CDC protocols and structures. Coverage metrics aid verification teams to set crisp sign-off targets for each phase of their CDC verification flows. This methodology ensures a high quality of verification and a strict adherence to standards.

Overall time and costs are saved by cutting time spent on the over-verification of certain aspects. This methodology eliminates the possibility of functional errors occurring later in the design cycle, by ensuring that all aspects of CDC problems have been covered comprehensively during verification. Using our proposed coverage models and metrics, verification engineers can achieve accurate and reliable CDC verification systematically—with measurable target goals based on application requirements.

6 REFERENCES

[1] Jean-Francois Vizier, Dennis Raemekers, Zheng Zhou, *Verifying clock-domain crossing at RTL-IP level using coverage-driven methodology*

[2] Chris Rockwood, Chip Design, *Keeping Time with Clock Domain Crossings in FPGAs,*

[3] Mark Litterick DVCon 2006, *Pragmatic Simulation-Based Verification of Clock Domain Crossing Signals and Jitter using systemVerilogAssertions*

[4] Doug Smit, John Aynsley, *A Practical look SystemVerilog coverage - Tips, Tricks and Gotchas*

[5] Ping Yeung, *"Five Steps to Quality CDC Verification"*, Mentor Graphics, Advanced Verification White Paper

[6] Chris Kwok, et al, *"Using Assertion-Based Verification to Verify Clock Domain Crossing Signals,"* DVCon 2003

[7] Tai Ly, et al, *"Formally Verifying Clock Domain Crossing Jitter Using Assertion-Based Verification",* DVCon 2004

Appendix A – CDC Coverage Metric for Case-Study Design

Step 1. Design Setup Validation

Sequential Cells	St	12352
Unconstrained flops (no qualified clock connected)	Su	232
Constant Clock	Sc	0
Constant Data with No Reset logic	Sd	0
Coverage (C1)	$\dfrac{St-(Su+Sc+Sd)}{St}$	98.1%
C1 Revised		100%

Step 2. Clock Domain Crossing Analysis

Missing Synchronizers	Scalar crossing no-sync	526
	Bus crossings no-sync	257
	Xm	783
Incorrect Synchronizers	Combo-logic before sync	3
	Mux-sel with multiple sync	1194
	Xi	1197
Good Synchronizers	Mux Sync	330
	FIFO	124
	DFF	47
	Handshake	316
	Xg	817
Total Crossings	Xt	2797
Coverage (C2)	$\dfrac{Xt-(k1*Xm+k2*Xi)}{Xt}$	29.2%
	Revised Xm	0
	Revised Xi	16
C2 revised		99.4%

Step 3. Protocol Verification

Synchronizer Protocol	Total Checks	Uncovered Checks	%age Covered
Mux Synchronizer			
Data stable check	330	27	92%
Tx_min_cycle check	330	23	93%
2 Flop Synchronizer			
Data stable check	47	2	96%
handshake			
Data stable check	316	4	99%
Req_has_ack	28	2	93%
Ack_had_req	28	2	93%
fifo			
write ptr is gray-encoded	14	3	79%
read pointer is gray-encoded	14	3	79%
overflow check	14	3	79%
underflow check	14	3	79%
Total Checks (Pt)	1135		
Total Uncovered Checks (Pu)		72	
Coverage (C3)	$\dfrac{Pt-Pu}{Pt}$	93.7%	94%

Step 4. Reconvergence Verification

Static reconvergence		466
Fixed structurally		295
Grey-encoding checkers	Rt	171
Uncovered Checkers	Ru	28
Coverage (C4)	$\dfrac{Rt-Ru}{Rt}$	83.6%

Step 5. Metastability tolerance

Paths Verified	Mt	548
Uncovered Paths	Mu	42
Coverage (C5)	$\dfrac{Mt-Mu}{Mt}$	92.3%

Overall Coverage	Original	Revised
$C = \ k1*C1 + k2*C2 + k3*C3 + k4*C4 + k5*C5$ $C = \ 0.3*C1 + 0.2*C2 + 0.2*C3 + 0.1*C4 + 0.1*C5$	74.5%	96.2%