The size and complexity of designs, and the way they are assembled, is changing the clock-domain crossing (CDC) verification landscape. It is now common for these complex SoCs to have hundreds of asynchronous clocks.

As CDC signals can lead to metastability, CDC metastability issues have become one of the leading causes of design re-spins. This trend has made CDC verification an even more critical step in the design verification cycle than before. Naturally, this requires more time and effort to verify CDC issues and poses multiple challenges for CDC verification at the SoC level.

- There is an urgent need to move beyond the traditional, flat CDC verification approach. Flat CDC runs on an SoC are performance intensive, time-consuming, difficult to debug, and tend to prolong the verification effort. The inefficiency of flat runs is exacerbated by a high level of redundancy in debugging as the same CDC bug may be replicated across multiple instances of the same module in the full chip.
- The increasingly common usage of third-party IP and design reuse shifts the emphasis of CDC verification. Often the IP blocks have already been verified to be free of CDC issues (i.e., they are CDC clean); thus validating the integration of IP into mega-blocks and mega-blocks into sub-systems becomes more important. To handle complex designs efficiently, we also require a good methodology and mechanism to hide the internal detail of the IP while still verifying the integration comprehensively.
- As more IP is integrated together, reconvergence of CDC paths is becoming one of the most significant concerns in CDC design and verification. To avoid metastability issues, synchronization logic is used on CDC paths. This synchronization logic stops the propagation of metastability but can cause unpredictable propagation delays. As each CDC path can have a different unpredictable delay, the convergence of these CDC paths can lead to functional errors.

In this article, we describe the hierarchical data model (HDM), which is the backbone of the Questa® CDC hierarchical verification solution. The HDM is equivalent to an abstract CDC model of the IP that captures the CDC intent of the block along with its integration rules. It is a generic data model that can be seamlessly reused across releases and across designs wherever the IP is reused. It can also be a performance efficient alternative to the traditional flat CDC verification flow.

**THE NEXT GENERATION HIERARCHICAL VERIFICATION METHODOLOGY**

We have developed the next generation hierarchical verification methodology in the Questa® CDC verification environment. It addresses the shortcomings of existing methodologies and the additional limitations associated with hierarchical verification. It deploys a systematic and accurate hierarchical verification approach that leads to faster CDC closure. The configurability and flexibility of this methodology ensure it can cater to CDC verification at different levels – from IP blocks to sub-systems to the entire SoC.

This new hierarchical methodology is based on the HDM. It has several important characteristics:

1. HDM is an abstract heterogeneous representation of a module. It is stored in an abstract object-oriented database in a compressed format.
2. HDM is lightweight and portable. It captures various information about a module; such as the CDC intent of the block, interface attributes, environment constraints, schematic information, integration requirements, and possible configurations.
3. HDM is compact and fast for processing without losing any CDC related information. It is designed to store abstract netlist representations to support CDC reconvergence analysis at the top level. Hence, for CDC analysis, it has significant advantages over other interface representations; such as liberty, design constraints, timing constraints, etc.

4. Even though HDM is a binary database, users can decompile it anytime to see the internals and also override the interface attributes with constraints. Figure 1 shows an example of a decompiled HDM file.

The basic flow of the proposed hierarchical methodology is illustrated in Figures 2 and 3.

**Hierarchical data model generation during IP verification:**
During IP-level CDC verification, a data model is generated along with CDC results. This HDM contains all the necessary information about the block to verify and debug issues during block integration in the SoC. The IP designer can choose to generate the data model for each configuration of the IP. Also the designer can embed the integration requirements in the HDM. For example, if a clock port is expected to be connected to a specific clock generator module, then such connection information can be provided.

### Table 1: Inferred Port Constraints

<table>
<thead>
<tr>
<th>Port Type</th>
<th>Group</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock Ports</td>
<td>G1</td>
<td>-</td>
</tr>
<tr>
<td>Data Ports</td>
<td>CLK1</td>
<td>-</td>
</tr>
<tr>
<td>Constant Ports</td>
<td>CLK2</td>
<td>-</td>
</tr>
</tbody>
</table>

### Table 2: Inferred Port Constraints

<table>
<thead>
<tr>
<th>Port Type</th>
<th>Clock</th>
<th>Scheme</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Ports</td>
<td>clk[0]</td>
<td>multi_bits</td>
<td>combo-logic</td>
</tr>
<tr>
<td></td>
<td>clk[1]</td>
<td>no_sync</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>clk[2]</td>
<td>no_sync</td>
<td>-</td>
</tr>
</tbody>
</table>

**Figure 1:** User-readable Version of HDM

**Figure 2:** HDM Generation During IP Verification
during HDM generation. The IP designer can also control the visibility level inside the IP. If the designer intends to provide only the CDC model and hide the internal connectivity, then the HDM can be generated accordingly. Once the HDMs are generated for the intended configurations, they can be archived or passed on to the SoC team.

**HDM usage in SoC verification:**
During SoC-level CDC verification, the SoC team only needs to include the HDM files provided by the IP team. All necessary information of the IP is extracted from the HDM and used in SoC CDC analysis, and all the CDC issues across the IP block are reported to the user. Also, the user is notified about any differences between the IP-level and SoC-level settings or constraints. The integration requirements specified by the IP owners are also extracted from the HDM and verified in the SoC-level CDC run. For example, if the clock port is not driven by the specified clock generator module then the violation will be flagged.

**ADDRESSING EXISTING LIMITATIONS WITH HDM-BASED VERIFICATION**
We have developed and deployed constraint-based model methodologies for many years. Based on our experience with the existing methodologies, we were able to design the new HDM-based verification approach explicitly to address their limitations.

**Parameterized IP support:**
In most cases the IP blocks have multiple parameterized configurations, each configuration having different functionality. For accurate CDC verification it is important to generate a hierarchical model with the correct configuration to be used in the SoC run. An SoC can also contain multiple instances of the IP with different configurations. It is the responsibility of the verification methodology to ensure that a hierarchical model with the correct configuration is plugged in for the correct instance. The proposed HDM-based methodology addresses this challenge by automatically selecting the correct configuration data model and alerting the user in case no match is found. This use model also enables the user to perform CDC analysis for each configuration of the IP and generate the HDM files, as shown in Figure 4.

**Accuracy:**
The primary advantage of the proposed methodology is the data model, which stores all relevant information to ensure accurate CDC verification at the SoC level without any compromise on debug. Any complex crossing that can be detected by flat CDC analysis will also be detected by this methodology. Also, the reporting and debug capabilities match flat run behavior. For example, if the IP block has multiple fan-outs and fan-ins inside the block, then the CDCs for each fan-out or fan-in will be reported accurately in this methodology. In Figure 5, the IP input port is connected to synchronizers in the clk1 and clk2 domains inside Block IP1, and it also drives the output port. Using the HDM methodology all crossings are correctly reported — two synchronized crossings and one unsynchronized crossing through the output port.

Through this methodology, complex divergence and reconvergence issues can also be detected. Figure 6 shows where a reconvergence issue spans across two different IP blocks. Such issues can be identified using HDMs of Block IP1 and Block IP2.
**Debug capability:**
In most existing hierarchical methodologies, debugging violations across a block is very challenging. In most cases, during SoC verification the IP block is shown as a blackbox in the schematic. The proposed methodology preserves the block schematic in the data model and displays it to the user. Thus the user can view the complete CDC path even if part of it is inside the IP block. Figure 7 shows an example of a top-level schematic with a redundant synchronization violation across two IP blocks, reported in the SoC-level run.

**Support for IP integration checks:**
The HDM methodology supports integration rule checking, which is necessary for IP-based flows. During CDC verification of the IP, designers can also provide requirements on properties that should hold during IP integration. The recommendations

---

**Figure 4: Use-model for Design with Parameterized IPs**

**Figure 5: Accurate Crossing Detection by the HDM-based Flow**

**Figure 6: Reconvergence Detected by the HDM-based Flow**

**Figure 7: Redundant Synchronization Across IP Blocks Schematic Using HDMs**
will be verified through the data model during CDC verification at the SoC level. For example, if an input port of an IP is connected to a synchronizer, then the user can specify that this port should not be driven by combinational logic after integration. Similarly if some port of the IP is expected to be connected to some specific module or specific clock-domain, the IP designer can embed these rules in the HDM. The proposed methodology will verify this property during SoC CDC verification and notify the user if any of the specification rules fail. This methodology not only ensures CDC-clean IPs, but also ensures clean integration of the IPs.

Figure 8 shows a DMUX synchronizer at the boundary of the IP block. The DMUX synchronizer is valid only when TXDATA and TXSEL are driven by the same clock-domain. The IP designer can embed this information through the “hier assume port” constraint, as specified below. The HDM methodology will ensure this rule is verified when the IP HDM is integrated in any SoC.

Configurable visibility inside IP:
This methodology facilitates generation and use of each HDM with different visibility levels. For example, during generation the IP owner can control whether to expose IP internals through the HDM or not. This ensures this methodology can be used for encrypted IP as well as when the IP owner needs to pass on the HDM with the integration rules without any visibility inside the IP. In such cases the block schematic will not be available during SoC-level verification, and CDC issues will be reported up to the IP block ports. Also if the IP owner has provided visibility permission, the SoC owner can control whether to use the IP internal information or not.

Constraints-based methodology:
The proposed methodology also provides flexibility to use a constraints-based hierarchical model. Sometimes the SoC-level verification starts before the block development is complete. In such cases, generating an abstract model for the IP is not possible. In such cases, IP designers can describe the HDM through constraints that can be used during SoC verification. The constraints can also be generated from an HDM file. So in this methodology, there is flexibility to choose the use model for both IP and SoC verification engineers.

Reusable CDC IP:
The HDM is essentially a CDC IP that can be archived and used whenever the block is used in any design. Generally, IP and SoC development happens independently by different teams, often using different releases or different methodologies for verification. Also, after its development, the IP can be used in multiple SoCs across different releases. Every time the IP is integrated, the hierarchical model does not need to be regenerated. Once the IP is finalized the HDM can be generated and archived and then reused across designs and releases.

Figure 8: Example of Integration Rules for IP Blocks
CONCLUSION

In this article we presented a Hierarchical Data Model that is equivalent to a CDC IP. The HDM can be generated for any IP block with multiple configurations and can be reused whenever the IP is integrated in any SoC. The HDM-based CDC verification methodology eliminates the risk of missing CDC issues at the IP boundaries when IPs are integrated at the SoC level. It also ensures the requirements from IP designers are verified.

The advantage of this methodology is that it meets the challenges of an IP-to-SoC verification flow and provides configurability to modify the flow as necessary. Because the HDM methodology has debug capabilities similar to a flat CDC run, it ensures much faster analysis of CDC issues at the SoC level.

The HDM-based verification approach is more efficient than regular flat CDC verification, addresses capacity challenges of large SoCs, reduces redundancy in CDC violations, and leads to faster CDC verification closure.
VERIFICATION ACADEMY
The Most Comprehensive Resource for Verification Training

32 Video Courses Available Covering
- UVM Framework
- UVM Debug
- Portable Stimulus Basics
- SystemVerilog OOP
- Formal Verification
- Metrics in SoC Verification
- Verification Planning
- Introductory, Basic, and Advanced UVM
- Assertion-Based Verification
- FPGA Verification
- Testbench Acceleration
- PowerAware Verification
- Analog Mixed-Signal Verification

UVM and Coverage Online Methodology Cookbooks
Discussion Forum with more than 9625 topics
Verification Patterns Library

www.verificationacademy.com