**Problem Statement/Introduction**

Errors are random in nature! Errors do not happen in isolation!
- This is true across all types of errors--Standard Logic Design-Under-Test (DUT) errors, Interface errors or Protocol errors.

Coverage metrics look good but misleading
- Most verification engineers end up exercising error scenarios in directed tests, as the testbench doesn’t support standardized strategy to inject errors.
- Functional and code coverage might seem good with this strategy, but DUT error handling is left unverified from a real-world scenario perspective.

Testbenches (TB) should be built such that they exercise errors in mimicking real-world—in a truly random fashion.
- A well-thought-out comprehensive strategy is required to build UVM Testbenches for Constraint Random Error injection, where errors are intermixed with good traffic, where the testbenches not only gracefully handle errors but predict the design error handling capabilities.

The problem here could be termed as “Sequence—Coverage Problem”
- Inability to identify driver bugs. Leading industry recommendations suggest that the Universal Verification Component (UVC) or Verification IP (VIP)/Driver packets be collected for coverage, as monitoring errors and uniquely identifying them is quite challenging. The major drawback of this method is the inability to identify driver bugs, as the methodology doesn’t necessarily guarantee that the intended error was truly driven. The problem here could be termed as “Driver—Coverage Problem”.

DUT error handling is unverified from a real-world scenario perspective
- VIP vendors do provide good error injection strategy but there is lack of standardized approach on handling those errors in the testbench. The problem here could be termed as “Error handling Problem”.

**Implementation Details—Diagram**

STD UVC vs Error aware UVC

<table>
<thead>
<tr>
<th>STD UVC</th>
<th>Error aware UVC</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCI</td>
<td>uvm_info</td>
</tr>
<tr>
<td>VCI</td>
<td>uvm_error</td>
</tr>
<tr>
<td>VCI</td>
<td>uvm_critic</td>
</tr>
<tr>
<td>VCI</td>
<td>uvm_success</td>
</tr>
<tr>
<td>VCI</td>
<td>uvm_critical</td>
</tr>
<tr>
<td>VCI</td>
<td>uvm_failure</td>
</tr>
</tbody>
</table>

**Proposed Methodology/Advantages**

The proposed “Dynamic Error Injection & Handling” is a novel and comprehensive UVM testbench strategy, which details on end-to-end verification techniques required to build a robust error aware and automated error checking testbench.

The proposed strategy details on techniques needed to be adopted for constraint random error injection, fine control for error injection, modifications required for drive logic. It also details on the importance of the monitor in being error aware, and automated check aspects of the testbench.

The proposed “Dynamic Error Injection & Handling” Testbench Strategy can be summarized as below:
- Errors intermixed with traffic—all the time, using Constraint Random Error Injection
- Continuous Error Monitoring and Graceful Error Monitoring
- Automatic Response Prediction that includes predicting Status Register updates, Interrupts, DUT Error Response Packet etc.,
- Automatic Handling for Interrupt Handling, Response Packet Scoreboarding etc.,

**Results Table**

The proposed UVM Testbench Architecture is scalable and has been successfully adopted for comprehensive verification in highly complex Serial Design IP Controller in addition to solving the quoted problem statements, few of the key advantages of adopting this strategy were quicker coverage closure, higher confidence in verification signoff and significant reduction in reported late or silicon bugs.

**Conclusion**

Given the new-age IP/SOC complexities, constrained random error injection and error aware testbench have become the core verification requirement.

“Dynamic Error Injection & Handling” strategy is novel, an all-inclusive, completely scalable and comprehensive testbench verification strategy that can help verify any design of any complexity in a real-world perspective, mimicking the errors good and error traffic.

**REFERENCES**