

<Specification>

[Document Subtitle]

RyanHunter

[Company Name]

1. overall
2. debugEnhancement
   1. main purpose

* create a new report mechanism that can easy for debug
* based on current UVM framework, tiny effort to use this new mechanism
  1. features supported
     1. easy embeding

Our goal is that this kind of plugin can be easily fitted in ENVs with at less modification as possible.

TBD

* + 1. configurability

TBD

* + 1. transaction flow tracing

Sequence/transaction flow is probably a very often case a verifier want to trace on debugging. On how the test issues a sequence and how the transaction is converted on a driver component.

* + 1. phase/objection tracing

UVM already supports phase tracing, we'll add objection raised/dropped information additionally.

* + 1. thread tracing

may need investigation on VPI.

* + 1. log recording based on component
* log recording will record this component's sprint() at build/configure/run this three phases by default, and it can be configured by user.
* can configure to catch specific id-severity pairs to record in the log file of that component
  1. strategy

With following circumstance we can think out a better way to trace those information.

* + 1. transaction flow tracing

(need investigation on uvm\_sequencer, TBD)

* + 1. phase&thread tracing

UVM has built-in phase trace, considering thread trace combined with phase trace.

* to display raised objections during phase tracing
* anyway to get blocks thread? may need lookup in the SV standard (TBD)
  + kind like it's feasibility by using VPI of SV, need investigation.
    1. objection trace
       1. proposal

recording the phase/objection state in a log file, users don't need to configure this, just a swtich on/off, if simulation's hang, user can check that log file to see the last phase/objection state. (kind of in run\_phase EXECUTING with xxx objections raised). So as to record the duration time in this phase, simulation time, and reality time (if possible, based on whether we can achive this. Using $system of SV to get the server time)

* + 1. component based log recording

probably need to enhance the report server and replace the default on in UVM src, by coreservice::set\_server. So that we can get report object information and change the default LOG actions.

1. reusable test

All tests mentioned here refers to UVM based tests. We aim to establish a mechanism that every test created obey this rule can be easily reused in any other verification level that follows the same rule. This methodology should suitable for any UVM based environment. And will take projects in my work as experiments.

create a simple APB interface examples

TBD

* 1. strategy

In 2 different level of environment, the driving UVCs are different, that's why a test cannot run on different environments.

To fully leverage the UVM/SV random feature, we consider a constraint based test. But that's not mandatory for reusing methodology.

Use virtual sequence&sequencer mechanism. Use same sequence to start in a virtual sequencer,