What is the quality of your DV environment?

Wayne Yun

Advanced Micro Devices Inc.
Markham, Ontario, Canada

www.amd.com

ABSTRACT

The quality of design verification (DV) environments is a rare topic throughout the life span of most projects. Issues like disabled checkers, insufficient stimulus, etc. will result in a lower quality DV environment, and more design bugs are likely to escape without notice. This risk is not reflected by most widely used metrics like functional coverage, code coverage and regression passing rate. The number of silicon bugs is relevant at this point, but it is too late. Synopsys Certitude offers not only an objective measurement of DV environment quality, but also the capability of finding weak spots of a DV environment. Once such weakness is fixed, more design issues could be identified and design quality can potentially be improved. This paper presents a real Certitude user experience of improving DV environment quality. It covers background, theory, setup, regression, analysis, actions and results. A further developing area is also suggested.

Table of Contents

Table of Figures

Table of Tables

1. Introduction

Functional verification is used to improve quality of design. Over decades of practice, many methodologies have been developed. For each methodology, a set of metrics are employed to measure verification process and quality of design. Critical decision, tape out or not, is made based on readings of these metrics.

Though newer methodologies and technologies are always superior to those of previous generations, bugs still escape from verification flow and result in re-spin. The ever-growing size and complexity of design is one of the main contributors. But post-mortem of these bugs, more often than not, hints about something was not done correctly in the design verification environment or plan.

For example, a checker was disabled while debugging a failure, then it was forgotten and submitted together with the fix. Such a later design change which would have triggered the checker is not flagged, and goes into silicon.

In another example, a scenario is missed from the verification plan so a corner case is not tested. Badly, but not surprisingly, the silicon fails.

All different types of DV flaws deserve a question: how well does the design verification environment do its job? In other words, what is the quality of the design verification environment? By answering this question, hopefully the quality of verification can be improved, and less bugs will move to silicon.

This paper discusses the quality of DV environments regarding their capability of catching design bugs. This measurement is a different angle from most commonly used metrics.

2. Finding a Metric for DV Quality

From a project execution perspective, the less metrics the better. Existing metrics should be examined before a new one is introduced.

2.1 Test Passing Rate

For verification environments adopting directed-test methodology, test passing rate is the most important metric. It is the ratio of the number of passing tests and total number of tests. It is a good progress metric.

From a DV quality perspective, it does not indicate if any design scenario is missing from the test plan, nor does it indicate any checking not done. The worst case is a test that simply prints ‘pass’ and does nothing. The test passing rate will show as positive, but actual verification is missing.

Test passing rate is not an indicator of DV quality.

2.2 Functional Coverage

Functional coverage is the most important metric for constrained-random methodology. It is defined based on verification requirements.

From a DV quality point of view, anything missed in verification requirements is also missed in functional coverage. Moreover, it only records the fact that a scenario happened. If design behaved wrongly, and checking is not functioning for any reason, the coverage point becomes a false positive.

Functional coverage indicates completeness of targeted DV stimulus, but not the correctness or completeness of DV checking.

2.3 Code Coverage

Code coverage can be deployed for both directed and random methodologies. It records design states and paths the stimulus drives it through.

If a scenario is not stimulated, there will be a hole in code coverage. In real projects, code coverage often is not driven to 100% because of resource reasons. Even if 100% is achieved, there still could be missing design feature, or incorrect DV checking.

Code coverage does not indicate correctness or completeness of DV checking.

2.4 Design Bug Rate

Design bug rate is typically the number of design bugs found each week. When it reaches zero or another low threshold, it is indicated that verification environment is close to its capability limit.

Design bug rate does not tell how many bugs the verification environment missed or hid.

2.5 Number of Silicon Bugs

After hardened in silicon, design bugs will be revealed throughout the rest of its life cycle. These bugs escaped from the verification process. The number of silicon bugs could be used as an indicator of DV quality.

The difficulty of using the number of silicon bugs as a DV quality indicator is obvious. When a silicon bug shows up, it is too late – a re-spin is needed to fix it. On the other hand, the number of silicon bugs is only a portion of issues escaped from the DV, especially in early use of the silicon.

The number of silicon bugs is more suitable for a conformational metric.

2.6 Calibration Weights

It’s not surprising to see that DV metrics aren’t indicative of DV quality, since these readings are engineered to reflect design quality. Furthermore, design is not purposely structured to find DV environment bugs.

This type of problem is not new – two objects interact with each other, A can measure B, but cannot measure A. This raises the question of how to make sure A has an accurate reading. In the example of scale and weights, calibration weights are used to get standard readings from a scale.

To fit into this model, the DV environment is the scale, design is the weight. Many known "calibration designs" are needed to check if the DV environment is up to the quality requirement.

The ratio of number of faults found by the DV environment and total faults is a good metric for DV environment quality. But many "calibration designs" are needed to make it statistically meaningful. Given the massive number of possible ways things could be done wrongly in design, a large number and also high quality of known defects are needed in the design. This is impossible to be achieved manually; it has to be an automatically generating tool.

3. Answer from Synopsys Certitude

Certitude Functional Qualification System injects bugs (faults) into RTL design. A design with bugs should cause the DV environment to flag it. For a simulation-based DV environment, at least one test should fail. If the DV environment fails in demonstrating this response, there must be a problem in the DV environment. The design with an artificial bug is the "calibration weight," and the "scale" is the DV environment – the failure to give a correct reading has to be fixed.

Certitude employs a statistical method while sampling the fault domain; it runs tests and calculates metrics that show the ability of the verification environment.

3.1 Bug Finding Model

The figure below is copied from the Certitude User Manual.

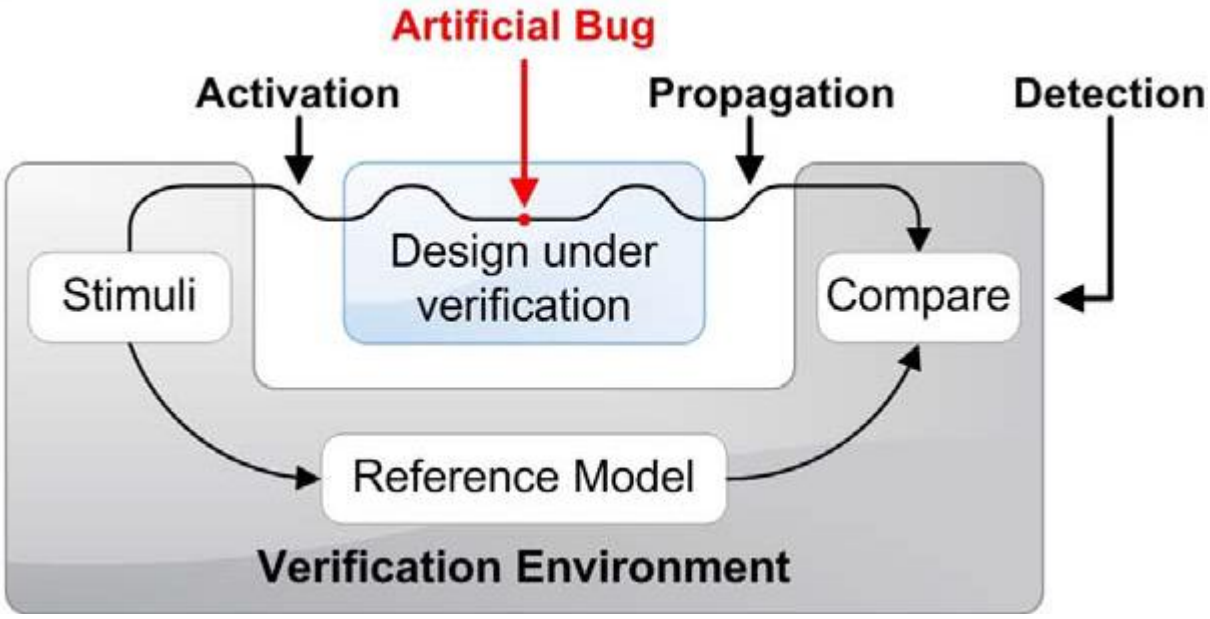

Figure 1 Certitude Bug Finding Model

There are three steps for finding a bug in any verification environment.

The first step is activation. Stimuli have to reach the bug and activate it, and the bug should generate its effect after it is activated. If the stimuli are not sufficient, the bug will not be activated and thus will not be found.

The second step is propagation. The effect of the bug has to propagate to an observable point, like an assertion, a probed signal or a pin which are monitored by DV components. If a bug is not propagated, it cannot be found.

The third step is detection. The DV code has to report an error based on erratic behavior caused by the bug. Otherwise, the bug is not found.

According to which step a bug is kept from being found, Certitude put its findings into three categories: non-activated, non-propagated and not-detected.

3.2 Certitude Working Model

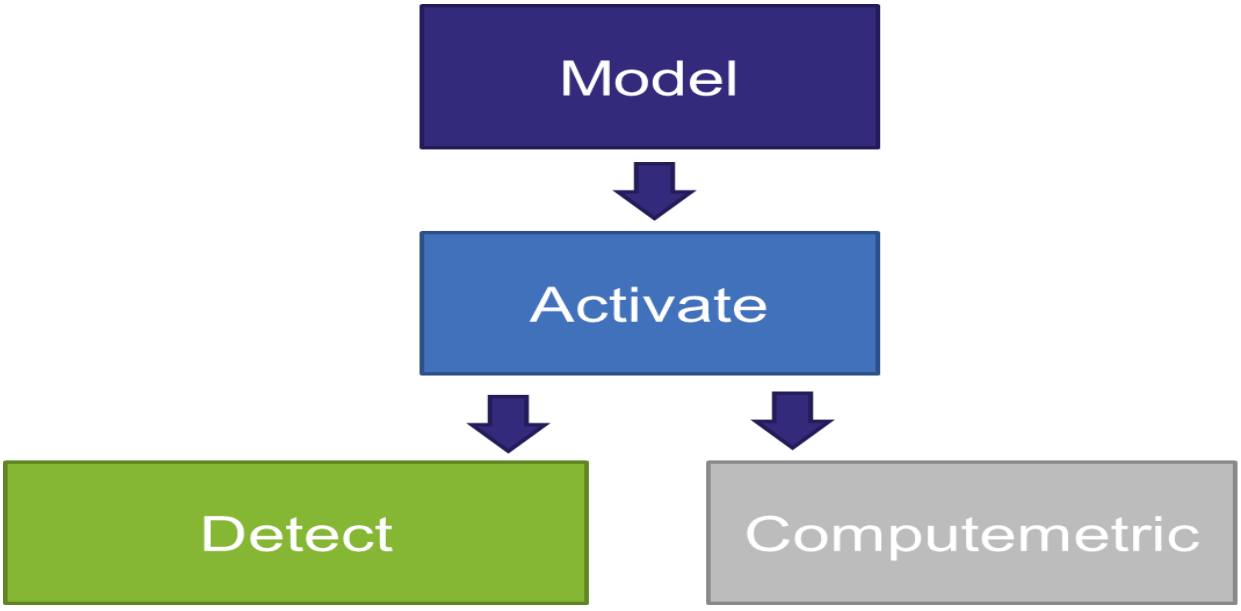

Figure 2 Certitude Working Model

Certitude runs in phases. The first phase is “model.” Certitude reads design files, analyses, optimizes and injects faults.

The second phase is “activate.” Full regression is run with Certitude instrumented code, and activation and propagation information is collected.

“Detect” can be run as the third phase. Certitude will inject faults in the design, and run tests most likely to find them. An HTML format report can be generated.

"Computemetric" can also be run as third phase. Certitude will sample the faults, run tests to see if the verification environment can find them, and estimate the percentage of faults the DV environment can catch. This percentage can be used as a quality metric.

3.3 Certitude Report

During the "detect" phase, an HTML format report having detection result can be generated at any time. It lists all faults by category and class. Faults are also highlighted in source code.

Class Name	Faults In Design	Faults In List	Non-Activated	Non-Propagated	Detected	Non-Detected	Disabled By Certitude	Disabled By User	Dropped	Not Yet Qualified
TopOutputsConnectivity	6	6	0	0	1	2	0	0	3	0
ResetConditionTrue	1	1	0	0	0	0	0	0	1	0
InternalConnectivity	0	0	0	0	0	0	0	0	0	0
SynchronousControlFlow	3	3	0	0	0	0	0	0	3	0
SynchronousDeadAssign	2	2	0	0	0	0	0	0	2	0
ComboLogicControlFlow	0	0	0	0	0	0	0	0	0	0
SynchronousLogic	1	1	0	0	0	0	0	0	1	0
ComboLogic	2	2	0	0	0	0	0	0	2	0
OtherFaults	7	0	0	0	0	0	0	0	0	0
All Fault Classes (9)	**22**	**15**	**0**	**0**	**1**	**2**	**0**	**0**	**12**	**0**

Figure 3 Statistics of Faults

3.4 Certitude Metric

The figure below is copied from the Certitude User Manual.

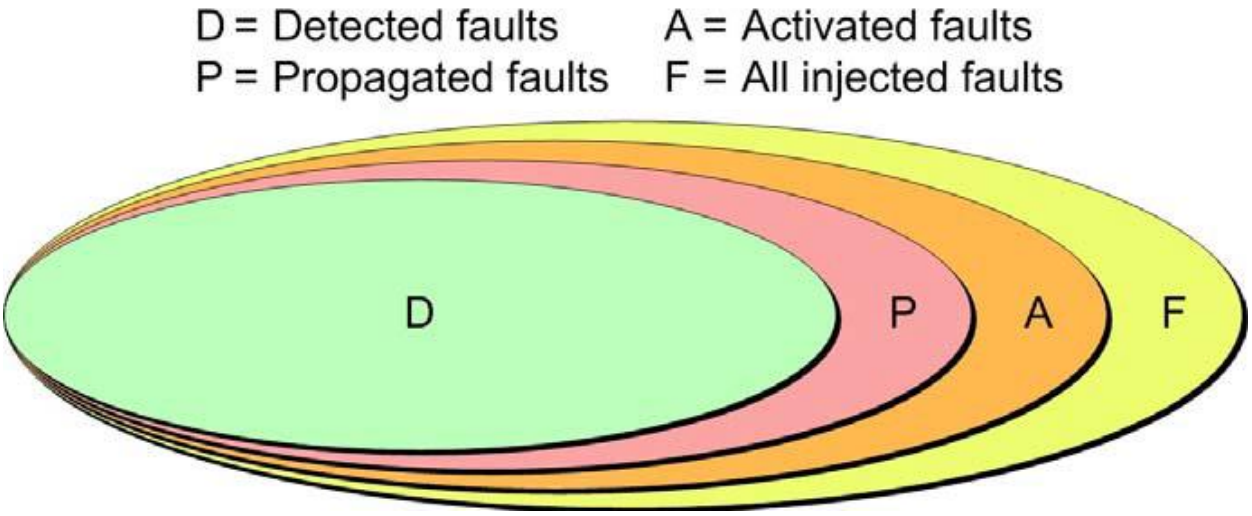

Figure 4 The Four Sets of Fault Status That Compose the Metric

Global Metric D/F measures how good the verification environment is at detecting faults.

Activation Score A/F shows how good the fault coverage of the verification environment is.

Propagation Score P/A shows how good the verification environment is at propagating activated faults.

Detection Score D/P shows how good the verification environment is in detecting propagated faults.

These metrics indicate weakness at different areas of the DV environment. Faults not activated are most likely caused by insufficient stimulus, or redundant logic in design. Non-propagated faults are result of dead design code, insufficient configuration to DUT, or not enough simulation cycles, etc. Those non-detected are more related to checkers in DV environment.

4. Running a Block-level Testbench

BlockA has about 25,000 flops. It has been taped out and in massive production for many years. Though new features are added for every generation, it is considered stable since not many silicon issues were found. Its testbench is in C, and its regression has about 1,200 directed tests. Certitude 2013.12p1 was used to gauge this verification environment.

4.1 Certitude Configuration

Certitude needs a configuration file certitude_config.cer. It is listed below with explanations embedded as comments.

```
# set simulator to VCS
setconfig -Simulator=vcs
# set top module name of design
setconfig -TopName=blockA
# No auto stop while detecting
setconfig -DetectionAutoStop=false
# Overwrite design files while compiling
setconfig -InstrumentOnTop=true
# Simulator is 64 bit, i.e. vcs –full64
setconfig -SimulatorBitWidth=64
# Do not prepare report in model phase, save turnaround time
setconfig -ReportPreGenerationInModel=false
```

4.2 HDL File List

File certitude_hdl_files.cer provides a list of all design files. Certitude can analyse the files and inject faults.

```
# Verilog HDL is compiled as SystemVerilog
# Faults should be injected to this file
addsystemverilog -file=/pathto/blocka.v –qualify
…
```

Phase “model” can be started with two files above.

4.3 Test Cases

Every test run by the DV environment should be listed in the test case file certitude_testcases.cer. and Certitude will run them.

```
addtestcase -testcase=testname1
…
```

4.4 Compile Script

Script certitude_compile is used by Certitude to compile the testbench. Essentially, it is a vcs compile command with Certitude and original options and file lists.

4.5 Execution Script

Certitude calls script certitude_execute with the test name to run each test. Results of passing or failing are given as feedback to Certitude.

```
# Command to run test
simv +testname=test1 …
# Pass test result to Certitude
if (testfailed) then
  echo "Test $1 failed"
  $CER_SIMULATION_RESULT -result=Fail
else
  echo "Test $1 passed"
  $CER_SIMULATION_RESULT -result=Pass
Endif
```

4.6 Metric of Testbench

With all five files, “activate” and “computemetric” can be run. Certitude chooses which faults are to be sampled, and decides how many of them based on error margin parameter. Default 5% was used. Which means error of Global Metric should be no more than 5% at given confidence level. After running about 28,000 tests sampling 300 faults, Certitude prints out metrics.

```
**** Scores ****

All percentages and margins of error are valid 95 times out of
100 (confidence level).

Global metric      (D/F) : 57.76 % +- 4.43 %
Activation score   (A/F) : 85.41 %
Propagation score  (P/A) : 71.87 % +- 4.89 %
Detection score    (D/P) : 94.08 % +- 3.21 %
```

From the above scores, it can be shown that this testbench is good at detecting faults, but not propagating. Overall it can detect about 58% of all faults.

4.7 DV and Design Improvements

The "detect" phase can be entered after "activate" or "computemetric." Certitude will try to find faults that the verification environment cannot detect. Table below shows wall clock time for each Certitude phase.

Activation	Metrics	Detect
4.6hrs	61.6hrs	192.0hrs

Table 1 Wall Clock Time for Each Phase

At "detect" phase, the first 5 non-detected faults took about 7 hours (wall clock time). When "detect" was done, about 3,200 CPU hours were consumed by tests. Certitude injected about 45,000 faults, and found 53 non-detected ones. While investigating these faults, seven RTL bugs were found and one DV bug was found.

5. Conclusions

Certitude can calculate quality metrics of DV environments, which are not replaced by other widely used DV metrics. Certitude metrics indicate areas of potential improvement in the environment. It can also detect faults not found by the environment. By fixing the weakness, more RTL bugs are found and this results in higher quality DV.

The best time to run Certitude is when DV environment is stable and before RTL freeze. In essence, Certitude is to find faults not activated, not propagated, or not detected. If it is run while stimulus or checkers are still under development, findings will mainly be in those yet to be finished area. It is not very helpful. On the other hand, if it is run after RTL freeze, cost of fixing RTL is much higher. The sweet spot is right before RTL freeze.

A new feature has been suggested for Certitude. A minor design change most likely happens after a bug is found. Is it possible to analyse the change and find which tests are potentially impacted? This smaller group of tests can be run, and turnaround time can be reduced. There could be two modes. One is to give Certitude the new file or the difference, and let Certitude to calculate affected logical cone. The other is to specify a design hierarchy which contains the change, and let Certitude to find the logical cone started there. As long the database was not removed Certitude will work in an incremental mode. If model is run again Certitude will recognize the code and faults are the same between versions and keep the same fault numbers. It will assign new numbers to added faults due to the changes. Certitude still parses all of the files. It is not incremental in that respect. However, this capability allows the user to keep the status for the unchanged faults.

DV improves quality of design by applying all types of stress. Certitude is the revenge of design. By measuring interaction between design and DV, the quality of both is pushed to next level.

6. References

[1] Datasheet of Certitude, Synopsys

[2] Certitude User Manual, Synopsys

[3] Certitude Reference Guide, Synopsys