Cogita by Vtool – A new approach to simulation debugging

Hagai Arbel

Abstract – This paper describes Cogita – a simulation debugging platform developed by Vtool. It describes a real-life case study that was conducted by one of the customers in order to measure Cogita's efficiency vs. traditional debugging methods.

Keywords – Simulation, Debugging, log-files, Data Analysis.

I. About Vtool's Cogita

Vtool has produced a new, next-generation debug solution named Cogita. [1]

Cogita takes an abstract, visual approach to debug that solves many of the issues of failure analysis and debug. It incorporates state-of-the-art ergonomic, visualization technology, makes use of psychological studies into problem-solving and large scale data analysis, and applies modern Machine Learning (ML) algorithms in order to examine large data sets in a cognitive manner.

Cogita has been proven to accelerate the debug process anywhere from 3X to 10X compared to the alternatives, log files and wave forms debugging or interactive software debugging.

Cogita can be applied to large-scale block verification, particularly tuned to operate with complex UVM testbenches. It also brings unique debug capabilities in tracking down intricate corner cases from SoC verification runs.

A. Improvements to the debug process

Cogita applies the following improvements to the debug process:

- Reduces the chance of taking an incorrect assumption for granted.
- Supporting the process of asking the right questions.
- Validating assumptions and answering questions faster.
- Revealing possible paths that are otherwise easily overlooked.

Hagai Arbel is with Vtool DOO, Milutina Milankovica 1G, Novi Beograd. E-mail: hagai@thevtool.com.

B. Key benefits

The key benefits of Cogita are:

- Accelerates debug time, up to an order of magnitude, for a broad range of complex bug types. Given that debug represents 25% of the entire development time of a semiconductor, this represents a huge resource-saving and time-to-market advantage.
- Improves design and verification quality over and above coverage assessment through visibility into verification scenarios, allowing a clear understanding of convoluted design code for easy team communication and cooperation.
- Extends debug for large-scale systems on hardware-software co-verification. [2]

II. Background and Problem Definition

Company-X develops complex ASICs and FPGA that facilitates massive traffic from CPUs to memory. The main module of the system (the DUT) and the UVM testbench are illustrated in Fig. 1.

Company-X verification team assessed from previous projects that such subsystem verification will take 10 months for 4 engineers.

Below we explain the main functionality of the DUT and testbench and the challenges in debug that Cogita, later on, was able to solve. The DUT, in essence, accepts READ/WRITE layered packets via the YBUS OB UVC. When these packets are aimed at the AXI4 interface on the right-hand side, they are called DIRECT. The DIRECT datapath is the simpler one but it has many complications on its own:

- Each read or write packet contains many fields that are later on translated to AXI4 bursts. The AXI4 UVC Slave responds to the packets and the DUT processes the response and sends out a YBUS response to the YBUS IB UVC.

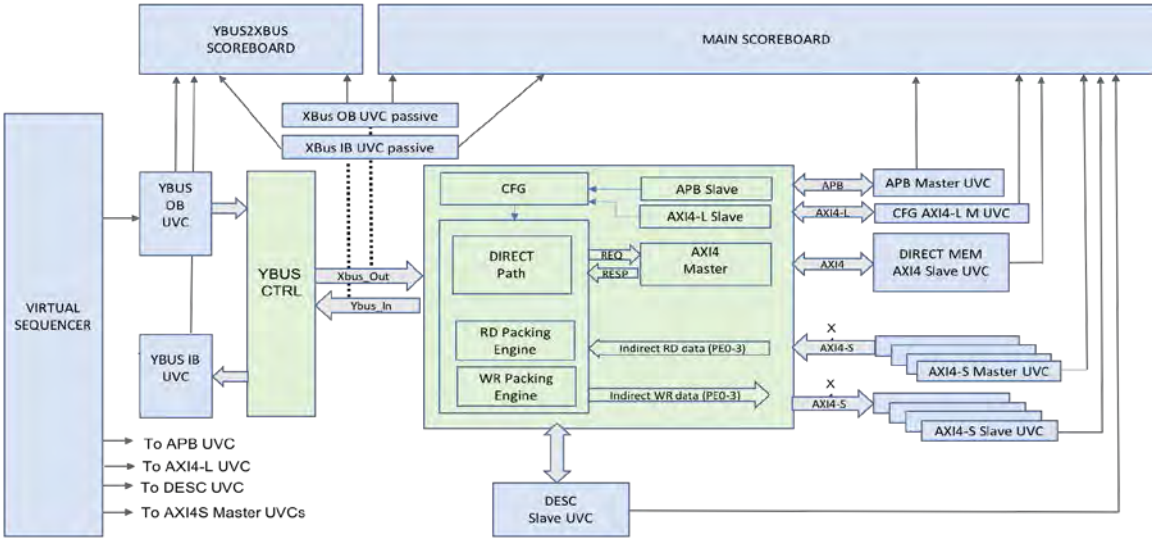

Fig. 1. The DUT and testbench

- There is a complex logic that transforms YBUS packets to AXI4 bursts. This also depends on the DUT configuration and Lookup Tables that are injected dynamically via the APB and the AXI4-L UVCs.
- The whole process is executed interleaved for 16 channels.

In parallel to the DIRECT path (and interleaved over the YBUS OB and IB interfaces), the INDIRECT datapath operated. This path is even more complex:

- In the WRITE INDIRECT path, the data that arrives in from YBUS OB UVC is layered to L4 packets, L2 packets, and YBUS Requests. Descriptors that are read from the DESC UVC, are telling the DUT how to pack the data and send it over the AXI-Streaming four interfaces.
- Similar mechanisms are used on the read side.

There is another datapath from the APB to the YBUS IB and back.

A fully random test in this testbench has a matrix of processed data that is almost impossible to track:

- Six different datapaths
- Three layers of packets and complex transform mechanism to/from the memory system.
- 16 parallel and interleaved channels.

Waveforms are almost useless in order to grasp what is happening in the test and in the testbench because they only show the DUT's signals. Without Cogita, the only way to analyze the sequences and the scoreboard is the log file.

In order to try and debug these complex tests, the team added a lot of UVM messages. But even then, they ended up with log files of 20 million lines or more, 2-5G of data.

Every failing test triggered many questions and challenges:

- What happened in this test? Which channels were active, which datapaths, what was the configuration at any moment?
- Was the layered data generated from the YBUS UVC legal? Did the descriptors interact well with the data?
- How to correlate the events in the waveforms to the log file?
- How to be able to track and analyze the data from the log, while still holding the bigger picture of the test scenario?

The team reported that a complex debugging session can take hours and sometimes days when you include the communication with the RTL design team. In some cases, a failure switched hand between design and verification engineers for a week until the root-cause (either testbench issue or RTL bug) was found.

III. Case Study

In order to test the efficiency of Cogita solving the above problem, the team conducted a case study.

A. *Saved player configuration*

First, they created a player configuration in Cogita that will show in one image the "story" of the test. Before the actual debugging of each failing test, this image was loaded in order to understand what happened.

Below are several examples of screenshots (Figs. 2 and 3) and the conclusion the team made.

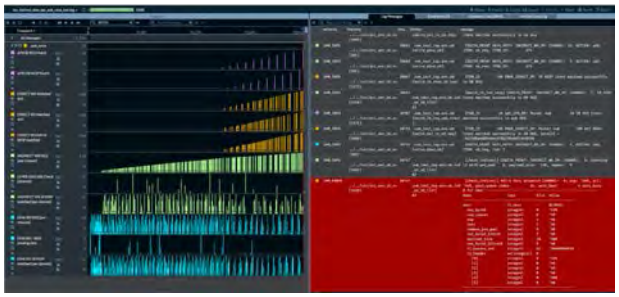

Fig. 2. Saved player configuration

- The test runs for 20,747 ns and then fails with 13 errors, the first one is INDIRECT WR datapath, channel 0, data integrity error.
- The APB datapath (purple players) is active both on request and response sides and has only 10 packets to that point.
- The DIRECT WR and RD paths (orange players) are active on request, AXI and response interfaces and have 108 and 109 items respectively.
- The INDIRECT WR path (green players) has 376 items till the failures, seems that all channels till 13 are active, AXI-S packets are checked properly till the failure point.
- The INDIRECT RD path (blue players) seems to operate well.

Since the failure is in the INDIRECT WR path, channel 0, the next player configuration (also saved by the team) provides a more relevant picture of the failure.

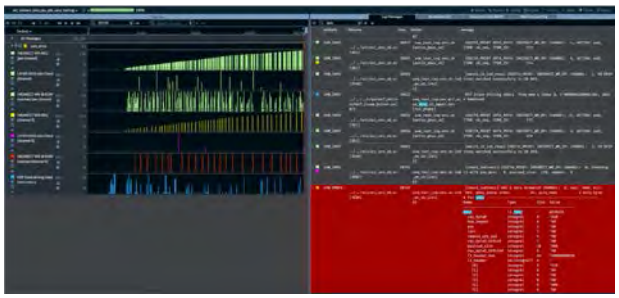

Fig. 3. Indirect WR path players

Now we can see channel 0 over the yellow player and that the failure happens in the third AXI-S check (the purple player).

B. Debugging a failure with Cogita

In this example, after a few good transactions, errors begin to appear, reporting data mismatch on the AXI port. This could mean wrong sampling by the AXI monitor, mismatch of expected vs. collected data in the scoreboard and actually, many other things. We do not know at this point.

UVM_ERROR ../../src/acc_env_sb.sv(1540) @ 1563 ns: uvm_test_top.env.sb.ind_wr_sb_list[#5] [match_ind_wr_axis] AXI-S data mismatch in byte 0 CHANNEL: 5, exp: 'h8043353c28db739f9a048e9f879c5fcd, act: 'h78efee8e652372f58403525e1363046f

Step 1: Open the log file in Cogita

Loading the log file into Cogita shows the basic view in Cogita (Fig. 4).

- Left side - A timeline with bars representing messages, the higher the bar, the more messages there are at this time. The red cursor is placed at the first error by default.
- Right side – Messages from the log, with the scroller placed in the time point the cursor is at.

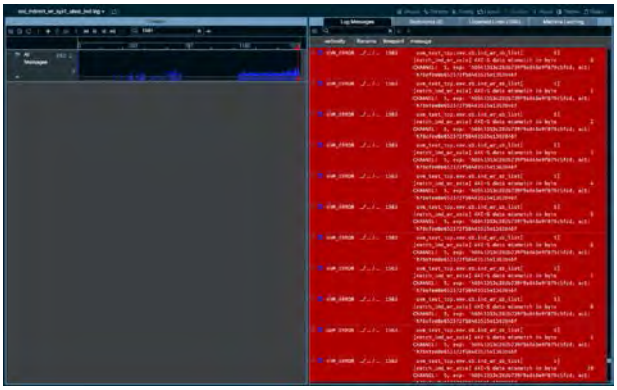

Fig. 4. Open the log in Cogita

Step 2 – Validating correct sampling by the monitor

Question: Is the reported "actual" value really the output of the DUT or is there a sampling problem with the monitor?

Answering with Cogita is easy because the engineer does not have to switch between viewing the log file in an editor and the waveform. Cogita can read data from multiple sources, waveform database included (vcd, trn, vpd, fsdb) and to jointly present them over a single timeline. Fig. 5 clearly shows the error message, the relevant signals of the AXI-S I/F.

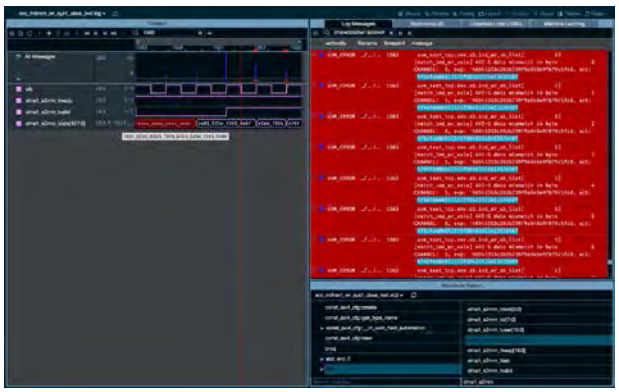

Fig. 5. Looking at waveforms

Answer: Yes, Is the reported "actual" value really the output of the DUT.

Step 3 – Looking for the actual and expected data at the input

Question1: Was the expected data sent over the input to the DUT and when?

Question2: Was the actual data sent over the input to the DUT and when?

Answering each of the above questions is extremely confusing without Cogita. One would have to search the log, for two different values, open the waveform viewer, look again, extracting the time point, etc. Not that it is impossible; It is time-consuming and more importantly, prone to the kind of mistake that send one back to the debug starting point.

Fig. 6. Looking at "expected" and "actual" players

With Cogita, we simply create two new "Players", as shown in Fig. 6.

Answer: Both expected and actual data were sent over the input. Actual (Fig. 6, Brown player) first at 1215 ns and Expected (Fig. 6, Green player) at 1267 ns.

Step 4 – Analyzing the data channels

So now, we know that both expected and actual data from the error were sent into the DUT.

Question: To which channel each data belongs?

To answer this question with Cogita, we set the Height of the bars to represent values of interest from within the message.

As shown in Fig. 7, each player's Height represents the channel number. It is clearly visible that expected data (starts with h'8043) was sent over the input with channel 5, while the actual data (starts with h'78ef) has channel number 4. Since the UVM_ERROR at the end has a height of 5 (it checks channel 5), we conclude that the DUT had switched the data from channel 4 onto the output of channel 5.

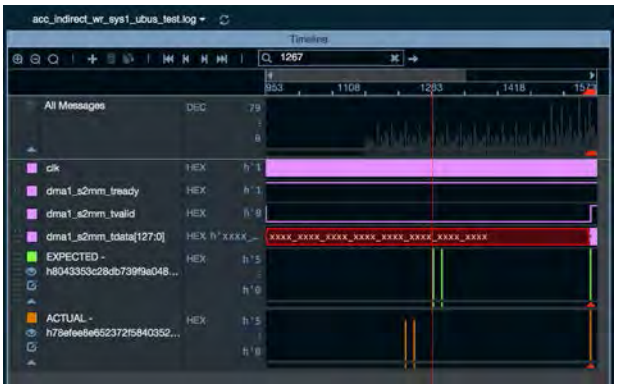

Fig. 7. Looking at Channels 4 and 5

Answer: Data h'78ef belongs to channel 4 but sent out over channel 5. Data h'8043 belongs to channel 5 and is lost. This is an RTL bug.

C. Debug process summary

In this case study, the debug process with Cogita included three steps.

Cogita helped the process in several ways:

1. Each answer leads to the next meaningful question.
2. Finding the answers (or validating the assumptions) is much faster.
3. The chances of taking a wrong turn, restarting the debug process all over, is almost impossible using Cogita.

IV. COMPARISON

This chapter shows a comparison between Cogita and two traditional debugging methods, both use Synopsys debugging tools. The first is the DVE waveform viewer and the second is Verdi, step-by-step debug.

The team had created the tables I and tables II-IV in order to assess Cogita's efficiency. Cogita accelerated the debug process in many aspects. When it comes to junior engineers and complex systems, Cogita has an added value of helping in understanding the system and the test scenarios.

TABLE I
PROJECT PROPERTIES

Category	Project attribute
Company	Company X
Design type	Packet Manager
Team	4 verification engineers, 3 designers
Device Under Test (RTL) size	25,000 RTL lines
Test Bench (System Verilog) size	100,000 System Verilog UVM lines

TABLE II
COMPARISON BETWEEN COGITA AND TRADITIONAL DEBUGGING TOOLS - DVE

Category	DVE Waveforms + Log Files (Synopsys)
Tool ramp-up	None - We assume everybody knows how to use them
Ability to understand immediately what the test generally did	Impossible
Time for the test case debug	Assessing over one hour
Debugging speed factor for senior engineer - On average	1
Debugging speed factor for junior engineer - On average	1
Helping communication within the team	Limited - attaching a waveform snapshot to the bug description

TABLE III
COMPARISON BETWEEN COGITA AND TRADITIONAL DEBUGGING TOOLS - VERDI

Category	Verdi step-by-step (Synopsys)
Tool ramp-up	We already knew this but it took the team a few months to learn
Ability to understand immediately what the test generally did	Impossible
Time for the test case debug	Assessing we would not use Verdi - It would not be efficient for this failure
Debugging speed factor for senior engineer - On average	1.5X
Debugging speed factor for junior engineer - On average	1.5X
Helping communication within the team	Limited - attaching a waveform snapshot to the bug description

TABLE IV
COMPARISON BETWEEN COGITA AND TRADITIONAL DEBUGGING TOOLS - COGITA

Category	Cogita (Vtool)
Tool ramp-up	Two weeks for the basic level. One month for very high efficiency
Ability to understand immediately what the test generally did	Very easy
Time for the test case debug	10 minutes
Debugging speed factor for senior engineer - On average	5X
Debugging speed factor for junior engineer - On average	10X
Helping communication within the team	Helps a lot - We used Cogita in bug reports and communication

IV. Conclusion

This case steady demonstrates Cogita's efficiency in finding bugs during pre-silicon ASIC simulations. By applying new approach to data analysis, it helps verification team finding bugs faster than ever before.

This entire debug session took 10 minutes of work. The engineer who conducted this had only a few weeks of experience using Cogita. The engineer assessed that it would take at least five times to debug this error without Cogita.

References

[1] Anna M. Ravitzki, Uri Feigin, Hagai Arbel, "*Between the Dialog and the Algorithm or Innovative Technological Narratives Leveraging the Idea of Authenticity in a Human Being. Visualizing log files enables intuitive comprehension of complex test scenarios.*" DVCON, Munich, 2017.

[2] Anna M. Ravitzki, Uri Feigin, Hagai Arbel, *"The Big Data Revolution - Beautiful Servant or Dangerous Monster?*" DVCON, Shanghai, 2018.