2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

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

Anna M. Ravitzki, Vtool, Paris, France (anna@thevtool.com)

Uri Feigin, Vtool, Belgrade, Serbia (urif@thevtool.com)

What is between technology and ethics? The ethical dilemma is created by the user's addiction to computer hours, and much like any addiction it creates a type of alienation, through the media. We each build a world of "I" for ourselves through technology, while technology itself is incapable of touching our psyche, and a complex relationship is forged between human and media. We become the creators of content, in which language and speech exist within this mythical space – a space where the dia-log and the multi-log become an a-log-arithm that produce this existentialism in which we live. We ask to create states of dialogs, where the medium takes active part.

Keywords—log file; debug; test case; teamwork;

I. Introduction

What is technology? Technology is a tool that promises the continuation of the human race. The "Techné" is the knowledge that serves nature in maintaining the human nature. The technological world in which we live has altered the social order, the language of the Internet has positioned man in a different dialog with the material world and with himself, a language in which the media shapes the epistemology. Technological development leads us into a completely different type of interaction between people – words and speech have become images of pictures and characters (letters), and the spoken dialog has become rare, replaced by the technology of typing on a keyboard or on a phone, and we ourselves have become the creators of content in which speech and language exist within this mythical space – a space where the dia-log and the multi-log have become an a-log-arithm that produce this existentialism in which we live.

What is between technology and ethics? The ethical dilemma is created by the user's addiction to computer, and much like any addiction it creates a type of alienation, through the media. We each build a world of "I" for ourselves through technology, while technology itself is incapable of touching our psyche, and a complex relationship is forged between human and media.

At Vtool, this perception has led us to tackle the psychological and philosophical narratives of what is identified as "modern technology", and we ask to create states of dialog, where the medium takes active part. Vtool expands the relationship between technology and epistemology, technology becoming a tool that serves authentic time, in order to ensure an existential life experience. Cogita is a revolutionary verification tool that, by using it, it produces authentic time, one which offers to reshape mankind who consumes media, with a post-modernist emotional perception, reverting and philosophizing on models of self-searching and questioning within life, in the quest for a new dialog with others. We seek modern technology that copes with the narratives of mankind's solitude in his need for a dialog, where the log exists as a dia-log and not as an addiction to the logic rhythm (a-log-arithm).

II. Technological Epidemics

The term Verification Engineer is a relatively new one. Many of us were born before this title existed. By definition, verification engineers spend their days validating new technologies, to ensure they work. Their task is not an easy one, having to actively engage day in and day out in a fierce battle against a cunning foe. This enemy is called a "Bug". This "bug" can disguise itself as a typo, or as a human error. However, when you explore its behavior, it should always be remembered that even its name implies an organic entity: a bug.

And this bug is contagious. It tends to infect the same areas over and over again. Some areas are more vulnerable than others. Certain hours of the day or seasons of year carry a higher chance of infection. We even tend to call some of these bugs "voodoo", because they seem to suddenly emerge and then disappear again back to an unknown universe. Cleaning up as you code, not leaving leftovers, putting things in order - all of these measures can help repel these bugs and eradicate possible "epidemics".

If we examine the toolset of today's "Technology Doctors", an absurdity becomes apparent: a person in charge of the most advanced technologies, some even not yet invented, spends most of their time hunting down invisible bugs using tools that were created more than 50 years ago.

III. Data, or the problem of debug

Verification is a technological tool for verifying technological innovations using a practical system. Without verification, there is no technology. An estimated 50% of the overall verification process is invested in bug detection and in analyzing its root cause.

Debug is the process of "**identifying** and removing errors" [1]. Any verification engineer will testify that the latter task is often the easier one. Identifying errors involves looking at data and finding the root cause of the failure. We like to think of this process as an intelligent one, requiring a qualified engineer with a degree and preferably years of practical experience in the field. Intelligence is defined as "...the **ability to perceive information**, and retain it as knowledge to be applied towards adaptive behaviors within an environment or context" [2]. Yet, the problem here is simply having too much information. A person, however qualified, can only be expected to perceive so much data of any type. This problem amplifies as the amount of data increases. Luckily, the limits of data perception has been approached before, with some life changing solutions.

IV. How We See Data

"Grasping not just the facts but their integration into a meaningful whole." Jens Zimmermann [3]

Paddington, 1863. The world's first underground railroad opened its gates in London. Everything was simple. One railway. A single train. Two stations. Everything worked smoothly. Everything was clear.

London, 1908. By now more than a dozen companies built underground railways. And there were numerous stations. Each company was independent. Some used coal, others electricity, or gas. Nothing was inter-connected. The network was a mess. Then, in one historical moment, all the companies agreed to put up the sign **UndergrounD** over their stations. And have one unified map for all.

At this point, the Underground map was a must. Every train line was charted. All train stations. Everything was superimposed over a geographical map of London to help orientation across the city. The river Thames flowed in between, with its streams and lakes. So as not to exclude anyone, the map's scale was very small to ensure every suburb was shown.

But the map was a disaster. The layout of the underground clashed with the geographics details of the city, and there was no room left for clearly labeling station names. In an attempt to include the suburbs, 80% of the stations were crammed into less than 10% of the map. Simply put, there was too much data going on. And to make things even worse, new stations were constantly being built and opened.

1931 London. Harry Beck was a former engineer at the Underground Electric Railways Company of London. With some free time on his hands, and a keen artistic style, he set about to sort out this mess, although by then the map was an accepted norm for well over twenty years. In his words: "Tidy up the tube map by straightening the lines, experimenting with diagonals and evening out the distance between stations." His most significant revelation was that people take the tube **underground**, meaning they do not care for the landscape, or the distance, or even the relative location. All they wanted to see on the map, as Beck quickly realized, was how to get from point A to point B.

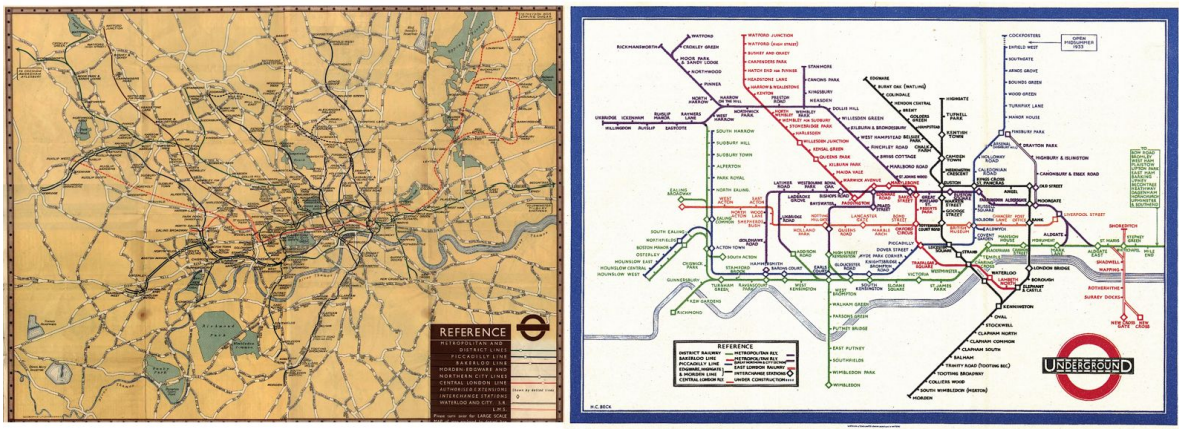

Figure 1 London Tube Map - comparison between the geographical and the schematical versions

Beck filtered out the relevant data needed to accomplish this task, and then also found the best way to display it. The outcome was a worldwide revolution. Even though he originally created the map for his own sake, in his own good time, it is still considered today to be a key design milestone, an icon of the twentieth century.

V. The Log File

A log file has a structure, where each line contains a timestamp, message, emitter. These log files may range from a few hundred lines to hundreds of thousands. No one can tackle these huge files, unaided, and the practice of printing out only specific parts of the entire data has become a rare art form. This is where the story of the underground map comes in. Before Beck's design, the tube map resembled a bowl of spaghetti. And now, to revolutionize the way we perceive a log file, we need to find a way that visualizes the test result without the clutter of geography, while maintaining the details around a bug scenario, without "compromising the suburbs". We need to highlight one data path, one class.

VI. Painting of a Log

"The purpose of analytical displays of evidence is to assist thinking. Consequently,… the first question is What are the thinking tasks that these displays are supposed to serve?" Edward Tufte [5]

When looking at a log file, the first question we tend to ask is "What happened?" But still, no one can read the entire log file. It is littered with unnecessary data, it is boring, long, tedious, and most of all highly time consuming. So, once we forgo the attempt to answer this question of "What happened?", and dismiss it as an impossibility, we are left with one option only: scrolling down to the very last log line, and tracing backwards from there, hoping that a few educated guesses will lead us through the murky swamp of endless log lines.

But what if there was a way to perceive the entire log file all at once, with the same immediacy we naturally grasp the world around us in one glance? As a verification company, we created such a revolutionary tool that does just that. We call it Cogita.

VII. Test Case A

To demonstrate our technique, we will walk through the debug of a real world environment. The device under test in our scenario is a bridge, with an AXI [5] master on one side and a memory interface on the other. The failure reported by the test is "RTL raised a request but scoreboard has no such request" (the scoreboard receives data from one end, matching and comparing to the other end). The AXI issues a burst of transactions, and these transactions are converted into single memory accesses. Multiple parameters need to be considered (start address, burst length, burst type), to check whether a certain memory location will be accessed. The problem here, is that multiple outstanding address phases are allowed (we can have 30 address bursts, and only then will data start to appear, with various possible data interleaving allowed). To put it simply, tracking is virtually impossible. We propose to display the entire test in a graph. The X-axis represents time progression, where left signifies time zero, and right is the end of the test, just like any wave viewer [Figure 2].

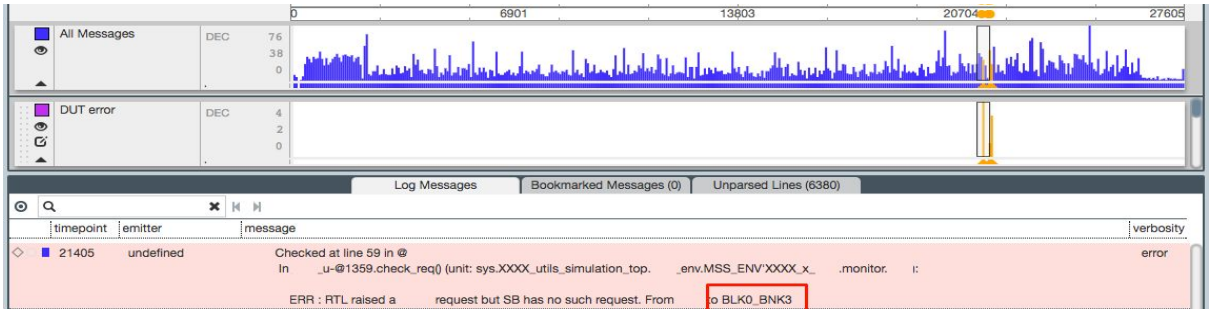

Figure 2 Visual representation of the log file on a timescale

The Y-axis, by default, represents the number of messages per time point. Upon loading the file, Cogita parses and displays the first error. In this case, it is apparent that the scoreboard is not waiting for any request to BLK0_BNK0. Therefore, the next step is to only display messages related to BLK0_BNK0 [Figure 3].

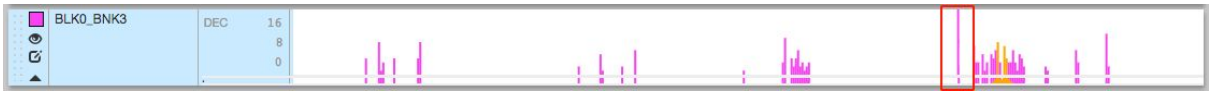

Figure 3 Representing messages related to a specific syntax (BLK0_BNK0)

By displaying only specific messages spread over the entire log, a clear image rises out of the complex array. The immediate visual anomaly that jumps out using this type of representation, is the tallest bar occurring a few pixels just before the error itself. We then continue by displaying all the messages printed at this time point, related to BLK0_BNK0 [Figure 4].

Figure 4 Filtering messages

Instantly, this filtered display shows us a single message type. It is apparent that this purple bar represents the address phase, with 16 data items to be expected. The green bars represent the data phase [Figure 5].

Figure 5 Displaying the address phase and Data phase

But - behold! We see that there is one more address phase issued here! Without Cogita, it would have been impossible to notice it, because there is a regular pattern most of us follow, where one address phase means moving on to the data phase, without further inquiry or delving deeper into the findings [Figure 6].

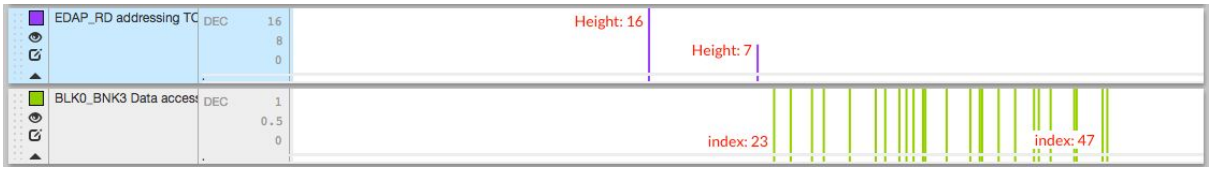

Figure 6 Zooming in and using the height value

Looking at the bars' height value, the first address phase has 16 messages, the second address phase has a height of 7, so a total of 23 data items are expected over a certain period of time, before the next address phase. By looking at the index number of the data messages, we see a total of 25 data items - two more than expected! A clear RTL bug. This debug took us less than five minutes, aided by an insightful visualisation tool. Without it, we would have spent well over an hour, to try and count all the requests from all the different sources, and more importantly, we would probably have completely missed out on that crucial second address phase.

VIII. Test Case B

Engineers gain many more capabilities for covering a wider range of tasks, which without Cogita's visualisation capacity would have been unavailable to them. For this test case, we will present the write address to an AXI slave, the "address" being a random indicator to start "painting" the picture, depicting a test scenario. We present a sample log line, with the phrase curr_addr found in a standard text editor.

 # UVM_INFO [monitor_axi_write_interface] @ 1950.0ns: => SKIP WRITE - curr_addr = 0x9a76d843, wdata[2] = 0x0 (beat_counter: 43)

There are 8360 occurrences of lines with curr_addr in this specific log file. Starting with the Y-axis, it represents the number of occurrences for the phrase 'curr_addr', per time point [Figure 2].

Figure 7 Representing the current address by occurrence

This one simple image [Figure 7] displays all the addresses written to the AXI slave in the test. It is immediately understood with just one glance. You can easily see that the AXI was active during three-quarters of the test. This one important detail cannot be exposed by attempting to read the entire log file itself.

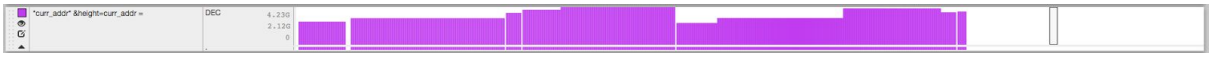

Figure 8 Representing the current address by value

Let us now define the Y-axis as the value of the address [Figure 8]. Using the altered display, a number of conclusions can be easily reached:

- You can see that the AXI only accesses addresses between the 2G and 4G address range.

- You can begin understanding where different transactions occur, according to base address jumps.

Figure 9 Representing the current address by lower bits

By focusing on the bottom 12 bits of an address [Figure 9], you can now see very clearly that all address access are incremental. Moreover, transaction boundaries can be deciphered intuitively from this image alone. By adding another layer, we display the actual transactions. Here we added another player displaying the AXI beat_counter, with the Y-axis presenting the beat in the current burst [Figure 10].

Figure 10 Representing the AXI write transactions by beat count

You can now clearly see every transaction in the test. They can be counted, while small, medium, and large transactions can be identified.

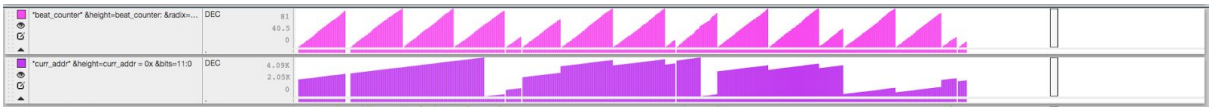

Figure 11 Layered view of transactions with address

By layering these two players one on top of the other [Figure 11] we can already see different regions of activity.

- You can see that the second to fourth burst continues the same address range.
- The fourth burst displays a wraparound of the address, once it reached the 4K boundary.
- Another wraparound can be identified in the tenth burst.

Figure 12 Representing the write data by value

By adding the data as another layer [Figure 12], we can see that it appears to have random values. The address seems denser in areas where bursts exhibit continuous address and wraparound. Clearly, something is going on here that needs to be looked into.

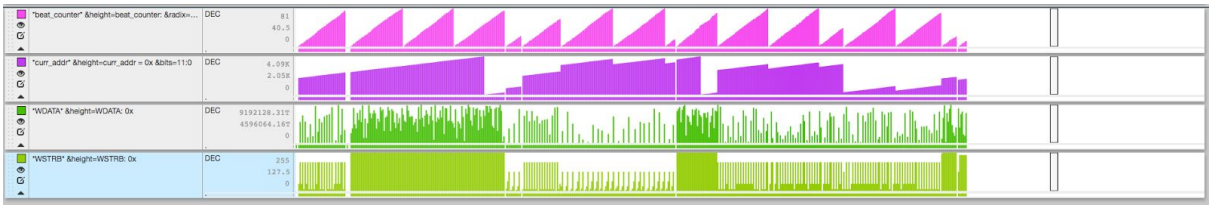

Figure 13 Layered view of the test scenario

Cogita lets you dig deeper still, and we can now also add the write strobe value [Figure 13], to display the areas that are written in every access. You can clearly see that the same areas of the continuous address range and dense data, have a full write strobe. This signifies a write to memory, as opposed to an access to a register in this test case. To sum up what we have learned from this one single image, which was created in less than a minute, and easily grasped by engineers that are not in-the-know of the environment details:

- The test has transactions from time 0 up to 75% of the total test running time.
- Seventeen AXI write transactions were initiated to the slave.

- Four short transactions, two medium lengths, and the rest with full length.
- All transactions were of incrementing burst type, with 4K wraparound at two distinct cases.
- The data written has no decipherable pattern and appears random.
- After the first transaction to registers, there were three continuous transactions to the memory with full write strobe.
- The memory was accessed three times during the test.
- There were no gaps between transactions for a significant amount of time.

And the list can go on and on… all this insight from just four players.

By using this innovative parsing and displaying technology, you can quickly uncover never-been-seen insight for any test scenario. These images are intuitively grasped and easily shared across all team members, eliminating the need of long whiteboard sessions or cumbersome email threads that try to explain the lingering question "What happened?"

IX. Benefits

We have targeted the use of this technique for analyzing log files as a risk-free process. A dedicated custom parser builder was designed to parse any given log file. Since Cogita is content-agnostic, any log file can be read by using the parser. This is not limited to UVM methodology, SystemVerilog, or any verification environment. From our internal work using this tool in real-life projects for various clients, we have proved that Cogita reduces detection time of complex bug scenarios by 40-50%, and standard scenarios by 15-20%.We define the bug presented in Test Case A as a level A transaction tracking failure. From our experience transaction tracking and pattern matching bugs are discovered within minutes using Cogita, regardless of the environment complexity. This often cuts debug time by 90% and more. In logic bugs, often related to FIFO logic or priority arbitration we have monitored a 20-30% decrease in the estimated debug time. Cogita has also proved useful in these cases, for tweaking the test scenarios to reach an expected outcome. Although not directly related to tracking bugs - this simplifies the entire verification cycle significantly, making it more effective.We are yet to substantiate improvement in connectivity bugs and registers mismatch. However, by visually displaying the test case, the three test projects showed that no bugs of such type were misidentified, meaning that no time was spent investigating an irrelevant route. This in itself also improves the efficiency of the overall process. Moreover, we have gained a much better understanding of teamwork scenarios, dramatically improving the communication between different departments, including designers, verification engineers, project managers, and clients.

X. Challenges

A few challenges surfaced while implementing the tool in several of our projects. The first setup time, for building the parser and creating the standard players, may take a few hours. To ensure this process needs to be done only once per project, export capabilities were added, which streamline the onboarding of entire teams after initial setup is complete. We also met challenges of changing the engineer's mindset. Every verification engineer is accustomed to long incomprehensible log files, and is used to printing as few messages as possible. Many environments are therefore littered with personal messages in the form of **MYNAME* syntaxes, meant to be found by a specific engineer. After using Cogita, all our engineers reported that their printing messages habit had improved, which carries much value in itself for higher efficiency.Our goal is to show that this tool revolutionizes the scope of work by employing a whole new way of thinking. We encourage the team to print as many messages as possible to the log file, and keep the message structure as simple as possible. When we say "no access was done to address 0", we must be sure we are not really saying "no print out to the log of a write to address 0 occurred". Also, we have seen the way new engineers embrace this innovative tool without really knowing how to use it even if they have not yet had a chance to practice this new way of thinking. We are approaching these scenarios by embedding A.I. that can analyze and suggest displayed patterns, eventually

allowing each user to see the entire image with a click of a button.

XI. Other Debug Approaches

In interactive practices, the simulation is run step by step, while the user can access all the design and testbench variable status, add break-points, etc. This is the main approach applied in most software debug processes. There are clear advantages running step by step as you check the behaviour of each logic part, because it enables the engineers to eventually understand the root cause of the failure. But, on the other hand, to fully grasp what really happened, the "whole story" must first be understood. The online approach, by definition, enables engineers to be clear on what is happening at one point, But the course of events that led to this point remains unclear and is often forgotten. Another problem is in the flow, making it very tricky to properly place a breakpoint. If you misplace it, everything has to be redone from scratch. Due to the many different things that are being triggered in parallel, there is no real sequential order of events, as there would be in a normal software. In offline approaches, for waveform debuggers, environment variables and events can be viewed, yet the tool has to be specifically configured for this. You can also create some logic operation between two "players" to get a 3rd player, but this is not easy and it must be numerical. Values will show up like a bus in the design, and not in bars, and the trend is not spotted visually. Current debug tools are trying to address this by combining the two. For example, recording the entire data base of RTL, or testbench variables. This affects performance, and users still have to decide independently which parts to record. After simulation is done, the user can then browse back and forth, using tracing mechanisms that contain a sequence of events, and not only the current state. The one key element that is clearly missing in every existing tool is visualising the log file.

XII. Future Development

The principles presented here are only the tip of the iceberg. By parsing a log file into a database, and extracting numerical values according to syntax - one can leverage numerous algorithmic techniques used extensively in other data science fields. The pattern discussed in Test Case A is a simple repeat pattern of request response. This pattern can often be tracked visually. By defining complex repeat patterns or growing patterns found in the test case, Cogita can automatically locate the anomaly and present it visually for quick grasping. By utilizing proven algorithms such as PCA and K Means, we can leverage pattern recognition to be automatic, and have Cogita present the anomaly at the click of a button. By mapping the topology of a verification environment we can leverage the knowledge gained from Graph Problems.

XIII. Epilogue

Technology becomes a tool that is meant to serve authentic moments, for achieving an existential life. We want people to have more time to live a creative life, and return to their natural human experience. As a verification company, our mission is to create an existential revolution that replenishes the idea of authenticity in human beings, through the use of innovative technological narratives. We have identified the debug process in general, and bug detection specifically, as the main bottleneck in today's ASIC and FPGA design processes. By approaching this issue from a standpoint that strives to expand the relationship between technology and epistemology, we appeal to the intuitive human capacities of pattern recognition, storytelling, visual essence extraction, and immediate visual perception of anomalies. We are facing a revolutionary era in debug, we call it the Cogita, Join us. The Authors would like to thank Dan Alexander and Hagai Arbel in their work and their participation in this article and the Cogita Project.

XIV. References

[1] Oxford English Dictionary

[2] https://en.wikipedia.org/wiki/Intelligence

[3] J. Zimmermann, "Hermeneutics, A Very Short Introduction," Oxford, 2015, p.

[4] M. Zachry and C. Thralls, "An Interview with Edward R. Tufte," Utah State University, Technical Communication Quarterly, 13 (4), p. 450, 2004.

[5] https://www.arm.com/products/system-ip/amba-specifications