The Big Brain Theory: Visualizing SoC Design & Verification Data

Gordon Allan
Mentor Graphics Corp
46871 Bayside Parkway
Fremont, CA 94538
gordon_allan@mentor.com

Abstract- Directing a team of engineering staff to design, verify and deliver a product as complex as today's System-on-Chip (SoC) devices is a risky business. That risk comes sharply into focus as the verification activity comes to a close and the investment in silicon production is finally initiated. There are many things we can do to mitigate that risk, but until some future date where we replace our human designers (or their designs), with a collection of nanobots, we have to make the most out of our human limitations of focus, motivation, innovation, objectivity and accuracy. Some of those limitations even apply to verification professionals!

We postulate that data is the key to success and provide concrete examples that illustrate the value of data. Collection, analysis and visualization of design and verification data is what we do, in our daily design/verification workflows, helped along by our favorite D/V methodologies and EDA tools. This author has observed that successful teams get more out of the data they capture and how they use it, and avoid being overwhelmed by too much of the wrong data or the wrong kinds of processing. Recognizing that value, and amplifying it, is the next step to process maturity.

This is not just another paper listing useful metrics from the 1990s – instead we describe the challenge of maximizing utilization of the best design/verification tool we possess - our engineering brains. Their performance is limited by the rate we feed them with meaningful data - data at the right level of detail or abstraction to solve the problem at hand, and ideally data which is presented in a fluid, visual form to make the most of our parallel processing superpowers.

We explore data visualization techniques which are becoming standard practice in other industries and which can be leveraged in ours. We must pay heed to the challenges of managing 'Big Data' in our domain, so that it works for us rather than overwhelming us.

Practical examples are described of both low-level and top-down visualizations representing situations that occur during a typical SoC design/verification flow, such as debug situations, or the need to view a complex design in a particular abstraction or aspect. Real examples are illustrated using the increasingly popular D3.js graphics library, providing rich visualization of Big Data.

I. Design/Verification Data

Data drives all of our engineering decisions, at least all the good ones. Especially useful is hard, scientifically-gathered and carefully analyzed data. That way we are more confident of its value, and more likely to trust it to make decisions on directions to take when defining our project, within our project or when migrating from one project to the next. Series of data add even more value than isolated data points, as they demonstrate consistency, become self-validating and provide trend and anomaly insight or cause/effect insight backed up by numbers.

In SoC design/verification engineering, our challenge is comprehension and analysis. How to make sense of and get the value out of data in a field that by definition is pushing the boundaries of technology and complexity with every project. It's a different kind of problem from measuring and evaluating a production line, or a scientific field study, or financial indices evaluation and prediction, or weather prediction.

However, we should first try to leverage the well-established techniques and tools used by those other industries.

We will discuss first the different shapes and sizes of data captured from SoC design/verification processes at low level and high level, then explore the varying degrees of 'magnification' that we need to interpret results from within different kinds of datasets and techniques we can employ to aid that search and zoom activity. Finally we look at ways to cut through the excess information and 'filter' to find the interesting data amongst the mundane. After all we are normally looking for data to validate our quality or completeness, by carefully looking for exceptions to the expected results. Most of our data analysis and visualization challenges use a combination of the above three things, summarized as data management, data scoping, and data filtering.

II. Data Management – Size, Shape, Dimensionality

Different levels of data exist and are used in any SoC design/verification project. There are many existing lists of common metrics that are used by chip design project teams in various ways. Rather than itemize the metrics again, we are going to look at the kinds of data, how they might best be presented to the engineers for analysis, extract some common themes and draw some conclusions on how EDA might contribute to productivity in this area.

One way to categorize data metrics is by dimensionality: data points or data series or trees of data or progressive snapshots of data. Depending on the shape of the data, different management schemes for capture/storage/analysis are applicable, and different types of visualizations are proposed, to maximize comprehension.

A. Static Design Data

The lowest level of captured data about our design and its verification consists of static data elements – values of individual design item metrics or configuration states. An example of this might be a pass/fail indication of a test or a regression test set, or an overall coverage number, or a power consumption estimate, die size estimate, maximum clock frequency of critical path, maximum packet or bus throughput across the chip, or any other design or verification 'index'.

Such data may be interesting but fleeting in nature, captured, consumed but not necessarily recorded for posterity. A suitable paradigm is a vehicle's speedometer. It shows a current value, and a unit of measurement. However a more effective visualization leverages our brain's ability and desire to read more into the data rather than treat it as a single point in time. What are the low/high limits? Is the current value excessive, normal, exceptional, remarkable in any way? What is the perceived rate of change – rising or falling, have we ever hit a danger zone, and so on.

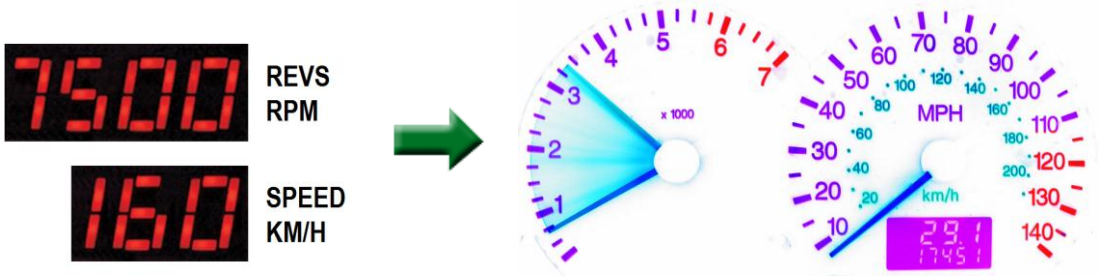

Figure 1. Example: Amplifying the value of 'static' data in context of change, rate of change, limits, warnings

Even a humble analog speedometer / tachometer can capture and display static data 'in context' to better enable the above kinds of subconscious analysis we are capable of. Replacing it with a simple digital readout of speed, or RPM, loses that extra value, and so modern digital implementations tend to preserve the analog gauge in some form.

B. Dynamic Data - The Temporal Dimension

The next level of data capture is to record static data values repeatedly across 'design time' i.e. simulated or emulated time, resulting in a data series representing unfolding or repetitive activity. This is where the traditional simulator / emulator output files fit in – sequential log files, and signal wave recording database files.

Other industries and endeavors have similar paradigms to SoC design/verification "waveform" views: EEG/ECG charts in medical applications, seismic data, financial performance over time, an audio engineers' multitrack mix.

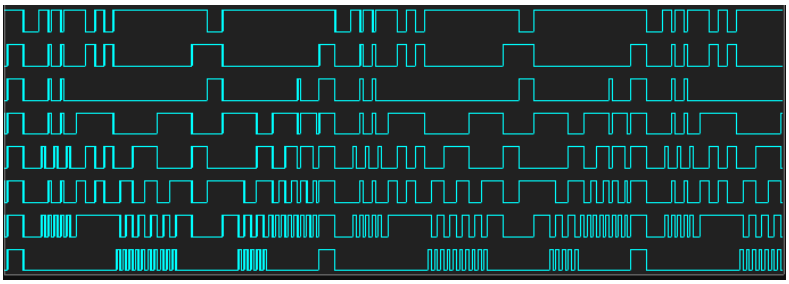

Figure 2. Example: Dynamic data flat presentation

Again our challenge is to amplify the value of a flat visualization like an ECG printout. The basic data tells the viewer something, but more detailed exploration, pattern matching, correlation with other data, or delving deeper into detail is where the value add comes for an SoC design/verification engineer, and so we can improve our visualization and interaction capabilities to keep up with the engineer's hunger for detail and answers.

C. Hierarchical Data – the Vertical Dimension

SoC designs and their testbenches are hierarchical in nature - gates inside cells inside state machines inside modules inside logical peripherals inside subsystems inside chips. Additional hierarchical overlays exist in many SoC designs: power islands and clocking hierarchies for low power, multilevel bus structures, and address decode hierarchies to name a few.

It is desirable to record data in a hierarchical form also, for two reasons: firstly to aid navigation of that data later when it is being analyzed, in the context of the corresponding design hierarchy node or level, and secondly to allow selective recording based on chosen hierarchical elements and depths.

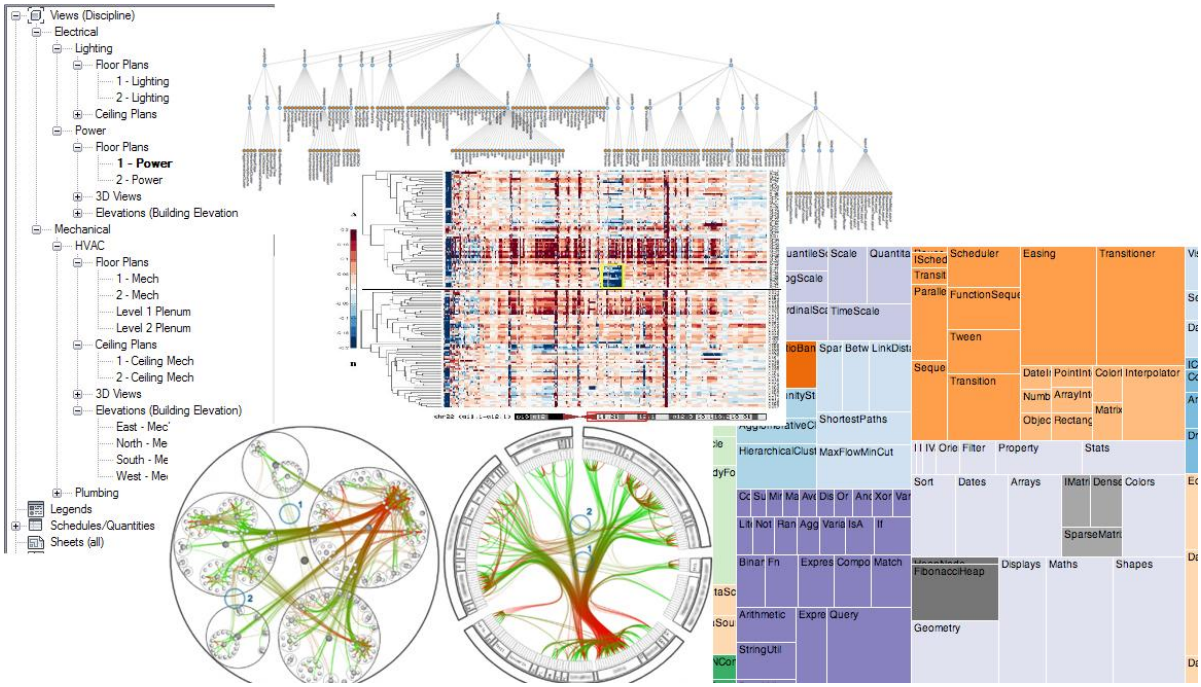

Figure 3. Examples of hierarchical data display and management

One aspect of hierarchy that may be particularly applicable to our industry but not necessarily to others: data recording comes at a cost, and so it is useful to have selective control over that – which levels, branches and depths of hierarchy are to be recorded. Recording the right data for the task at hand, and thus avoiding the necessity to rerun a costly simulation or emulation job, is a goal. Smarter data visualization can reflect this contradicting requirement: engage the user with the full potential data set, the full hierarchy, even though only partial data is available.

D. Data Across Time - The Historical Dimension

Although the most common data is recorded from design dynamic activity over 'design time' there is also a second temporal dimension – that of historical data. Data recorded across multiple runs taking place at different stages in the project's evolution towards completeness.

The point of this kind of data is to allow trend comparison across project lifetime and indeed across multiple successive projects. This dimension reflects 'real time' and normally proceeds in one direction: from project inception to definition to development to release (tapeout), and subsequent vectors to the next project. Data which is intended to be analyzed at this macroscopic level is normally a summary of individual metrics from snapshots of the design.

Rather than being a measure of 'my design' this kind of data metric and presentation is a measure of 'my project' and so may be useful to those in project leadership roles. There are opportunities to link this high level detail down

to lower level details or data that clarifies a situation or a 'hole' that needs investigation. Typical engineering validation takes place at this high level of data visualization just as it does with low level signal and activity debug, and the same principles apply: capture valuable data, amplify its value by analysis and presentation and interaction which relates it to other kinds of data. And repeat until project is done. Ideally, repeat across projects. Some of the most successful chip design startups are headed up by individuals who know the value of data and mapping it from one project to the next, or even one startup to the next. Being able to predict SoC design/verification activity timescales and the kinds of risks that can be encountered, from hard historical data, is valuable to the next project.

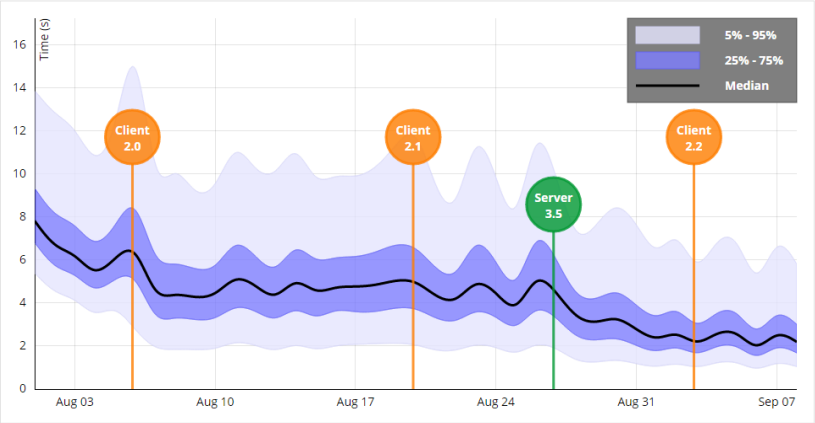

Figure 4. Historical data – snapshots of metrics and trends within a project or across projects over time

III. Data Visualization - Distance and Focus

This author has observed over many years and many SoC project teams, that there are two kinds of brains in the design/verification engineering profession. For lack of any psychological or physiological knowledge on the distinction, we categorize them here simply as 'bottom-up' and 'top-down'. The bottom-up brains can analyze details, absorb multiple strands of detail, deduce cause and effect relationships, and build from there until they reach a conclusion. Some engineers can zoom in on some detail signal transition data, and spot a bug or anomaly with ease.

The top-down brains on the other hand are good at absorbing the big picture, taking in a hierarchy of data from 10000 feet, detecting patterns and deviations and trends and multiple pathways through the data, before drilling down into targeted areas of detail for further analysis.

The main observation is that both of these types of skills are essential for large scale design/verification, that it is not generally possible to find both those engineering skills in one individual brain. Most importantly, the most successful teams seem to consist of engineers of both types, complementing each other, and good teamwork principles to enable that symbiotic relationship of the two.

And so it is with data visualization: reflecting the different kinds of engineering brain, our design, verification and analysis tools should provide the right kinds of visualization, interaction, and pathways through the data analysis process to accommodate those two types equally and provide maximal productivity boost to both types. And, our tools should encourage and facilitate the kind of teamwork and synergy that enables multiple brains to apply themselves at full potential to the problem at hand.

General principles to follow here are simple but often overlooked or only partially implemented as a favor to the user. These should always be implemented:

1. *Where more detail is available below this data item, announce its existence and invite elaboration*
2. *Where more context is available above this data item, announce its presence and invite exploration*
3. *Where relationships or linkage exists between two or more data items, announce their existence and invite investigation*
4. *Where membership of a class of item is denoted by an attribute, denote that clearly and invite filtering, searching, and association*

Depending on the ‘distance’ between the user and the data, various kinds of visualization ‘instruments’ may be required, to help the brain operate at the correct scope, and best utilize perception to read the data and extrapolate the hidden value.

Paradigms from other industries and endeavors are plenty and are embodied in the different instruments we use to see things around us.

We use a *microscope* to see tiny detail and focus on a particular aspect of biology for example, that same focus and zoom factor is doing two jobs: bringing the required level of detail into sharp visibility, and filtering out unwanted information that is either larger or smaller, or obscuring the detail in some way.

We use a *telescope* to bring distant data nearer, while looking at the heavens we track across the sky to the desired point, aided by commonly known artifacts as ‘signposts’, or by absolute co-ordinates. We adjust focus to see into the distance, in doing so also looking into the past.

We use a *zoetrope* and/or *rotoscope* to visualize animated imagery, presenting our brain with information that it knows how to process very well: motion, or the illusion of motion, leading to a realistic experience.

Figure 5. Tools for more effective visualization of data in the world and universe around us.

Each of these different kinds of optical instruments have parallels in engineering data presentation and evaluation, and we can learn a lot from centuries of scientific progress to ensure we do not regress in our own industry or accept only partial productivity gains by presenting data in an overly basic manner.

Some of the data metrics and debug data we capture and analyze needs microscopic analysis. What are we looking for? We do not necessarily know, but when we see it, we will know. Our brains will detect an anomaly, or something interesting that has not been seen before, or a pattern.

IV. Data Filtering – Distractions and Indicators

One interesting aspect of ‘focus’ in the optical visualization world is that is a filtering strategy – bringing data of interest into focus implies leaving other data out of focus. In the world of photography we add the creative aspect of ‘depth of field’ which is a selective application of focus across a range of distance which brings the subject of interest into view in a particular juxtaposition with other elements in the shot. The end result is an ensemble that is taken in by the brain as a whole that is greater than the sum of its parts. A non-flat representation of different data. Again, the science behind why this is effective lies in the way our retinas and brains process data, focus, motion. We have evolved that way to quickly identify food, threat, danger and make rapid deductions and reactions.

These principles can be leveraged when we are deep in debug of SoC design data and one particularly important principle is that of context. We are better at interpreting data when there is sufficient context, and furthermore in higher dimensionality data when we move from looking at item A to item B, we are better at correlation and rapid comprehension if that context is somehow preserved. A quick switch from view A to view B without context or motion, confuses our brain because in the real world we turn out attention from A to B in an analog manner, shifting plane of view, shifting plane of focus, depth of field. Maybe we are even distracted by something in the background during the transition, something that could be important

There are bad distractions (requiring filtering) and good distractions (requiring smooth context switch and motion when navigating the data set). Some general principles in this area are worthy of mention and discussion:

1. *When switching context vertically, horizontally, or via linkage, preserve sufficient context of the origin to reinforce the relationship*
2. *When elaborating more detail on an item, retain sufficient context to reinforce the membership that detail has of the original item*
3. *When ascending the data hierarchy, retain sufficient depth of open detail to link the two levels of data with a degree of persistence*

Simple visualization tools are available to achieve these goals in UX design: use of color, grouped items, spacing, and borders. Each has their place in maintaining coherency while transitioning context, and minimizing distractions or otherwise losing the viewer's concentration and hence losing the opportunity for analysis of the meaning of the data.

Careful use of motion and animation adds a lot of value above simple grouping and designation. While we have all seen 'bad' examples of overuse of animation, it does serve a purpose.

V. SCALING THINGS UP - BIG DATA

As the amount of data being gathered and analyzed increases, some patterns emerge. The problem space is not simply about the volume of data measured in Mega/Giga/Terabytes. While most SoC design/verification data challenges are only marginally large enough to warrant the kind of Big Data processing techniques that inherently unpredictable (i.e. human or geographical scale) domains require, it is still useful to observe the state of the art and learn from Big Data techniques where possible.

IBM [1] observes the "Three V's of Big Data Analytics": Volume, Variety, and Velocity.

Volume refers to the size of each transmitted/ processed/ consumed item of data, whether measured in records, transactions, or terabytes. There are obvious challenges with increased data volume, not just physical storage capacity, but accessibility of the right data with the right latency for the analysis problem at hand. Smart caching schemes and parallel storage and processing are becoming commonplace in many industries that have to find quick answers by trawling large data sets.

Variety of data at this larger scale may be more diverse, leading to more complex relationships and degrees of structure and indexing possible or desired in order to get value from the data. Big data is not "new" but organizations that have hitherto practiced the "hoarding" of data are now realizing the value and are turning to analysis, which means they need to think about how to correlate a great variety of disparate data sets.

Velocity is the rate at which data is being thrown around, generated and consumed. It may be at the near real time end of the scale, or batches of historical data built up over a longer duration. Again, technology can be applied in different ways to get the data feeds from A to B, i.e. out of A quickly so that A can get on with its normal function and produce more data, and stored in B where it can be later analyzed in the required timeframe. The generation timeframe and analysis timeframe may be the same, or different.

Innovative data presentation, filtering, correlation can be applied more, when there is a huge amount of data. On the other hand, some organizations become overwhelmed if they have too much data, suspect that there is high value information to be had from processing it appropriately, but do not know where to start. Ironically, a single human brain needs to comprehend the magnitude of the problem, how to break it down, even though they cannot comprehend the magnitude of the data without it being broken down in a meaningful way or to introduce a new dimension. Predicting weather is all about crunching huge geometric data sets, then looking across the time dimension, correlating history and predicting a trajectory, then presenting it in context of the existing data trajectory. Only then does it have meaning easily conveyed to the consumer.

With the increased use of hardware emulators and FPGA prototyping platforms as accelerated simulators, there is more emphasis on Volume, Variety and Velocity in our data capture and processing challenges. Our industry still thrives on "throwaway" data and does not lean enough from cumulative / historical data. That tipping point has yet to arrive but with current SoC design Emulation capacity [2] it is not far away.

The IBM [1] surveys suggest that use of Data Visualization "will grow faster than any other Big Data Analytics option". The reason for that outcome may be the same as the central premise of this paper: our brain is the best analytical tool, provided it is fed with the right stimulus.

VI. LEVERAGING THE WEB - THE D3.JS VISUALIZATION LIBRARY

The most popular personal computing application is the HTML web browser. It has also become the most ubiquitous tool ever created, with over 2.92 billion users in the world, and is fast becoming the most popular medium for presentation of information, with over 4.36 billion HTML pages in existence, comparable to the 129 million printed books which have been published, in the world, ever.

The modern combination of HTML, CSS, SVG (Scalar Vector Graphics) and Javascript enables a compelling solution for visualization of the written word, images, and graphics, especially when interaction is desired.

To enable rapid development of rich and flexible data visualizations we recommend a data charting or visualization library is used. The library we use here is called D3.js and is a JavaScript solution for manipulating HTML documents based on data sets, to translate that data into a graphical representation.

Like the popular jQuery library, D3.js employs a functional, declarative style, which reduces dramatically the lines of code needed to perform powerful document / data creation or manipulation operations:

```
var paragraphs = document.getElementsByTagName("p");
for (var i = 0; i < paragraphs.length; i++) {
  var paragraph = paragraphs.item(i);
  paragraph.style.setProperty("color", "white", null);
}
```

```
d3.selectAll("p").style("color", "white");
```

Example 1. Traditional Javascript and D3.js versions of a document modification function

Unlike many charting libraries, D3.js does not actually provide pre-canned charts. Instead it provides a high performance binding between data (normally in JSON form) and a dynamic HTML/SVG DOM (Document Object Model). This lets us code a mapping between the data and the lines, points, characters and other SVG elements on the HTML canvas that represent our choice of visualization, adding controls for axis scaling, drawing normal chart decorations, drawing data, highlighting, interpolating, smoothing, animating and, most importantly, interacting.

The pre-canned charts are still available, however, for developers to build solutions rapidly. There are dozens of D3.js example patterns [3] and [4] which provide rich data visualization and interaction templates as starting points for coding an application.

A. Comparison of D3.JS with charting libraries

There exist many alternate solutions for charting and data visualization in a more pre-canned manner than D3.js can offer, such as HighCharts and other excel-spreadsheet-like solutions, and of course there are advantages and disadvantages to both approaches.

The D3.js option requires JavaScript programming knowledge, specifically: OOP JavaScript styles and jQuery function calling style. The skills required are different from the average Perl/TCL scripting skills of CAD professionals, different from the VHDL / Verilog modular/algorithm coding skills of RTL design professionals, and different from the OOP SystemVerilog skills of the verification professionals. To use D3.js (or any presentation layer that requires JavaScript / jQuery), expert software comprehension is required. However, a lot can be achieved with a small amount of code, provided that expertise is available along with the resource to maintain the code from project to project.

Figure 7. Pre-canned charts e.g. from HighCharts widget library

The principal advantage of pre-canned chart widgets is that they can be created quickly and the code base is 99% maintained by someone else – either a purchased or an open source chart widget library. A resource with basic JavaScript/JSON/CSS knowledge can invoke such a chart widget on their HTML/CSS page with a tiny amount of JavaScript required. In most cases the behavior is specified by the data structure and a few options. The amount of implementation work with D3.js is higher, maybe 20% higher, but there are a huge amount of example implementations out there that are in the public domain to provide ideas and starting points for widgets. This closes the gap significantly with the pre-canned packages but lets us tailor them to be fit for purpose, providing the JavaScript expertise is available. Another aspect of the benefit of pre-canned charts is that wider browser compatibility is likely.

The principal advantage of D3.js is that the widgets can be highly customized and made interactive and responsive, scaling to huge datasets, and providing domain-specific data exploration and relationships beyond simple data series, within one chart. This allows the coder to capture the typical use case of a particular data visualization situation, to enable the user to access the detail they need, and follow the clues from one representation of the data to the next, to answer the typical questions they may ask of that data: correlation, filtering, cross-referencing.

There is also a performance benefit, because the user code is invoking most of the SVG drawing code directly, with the D3.js library providing superstructure and orthogonal code, rather than being inline and therefore slowing performance. However, performance should be viewed as a nice side effect; the main benefits are the features and flexibility.

B. D3.JS Data Visualization examples

A simple chart of data across multiple configurations or use cases can be brought to life by using D3.JS

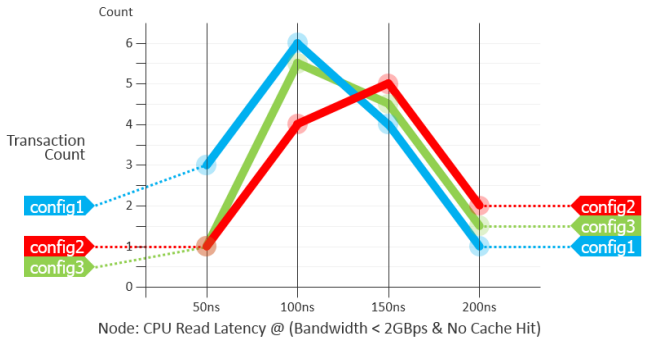

Figure 8. Simple HTML+SVG+CSS chart implemented by D3.js

The library provides for mouse hover behavior, animation between views, clear selection of multiple data sets the way the user wants to visualize them.

It is also possible to add multiple arbitrary overlays to show related data, or data limits, or validation of data points, in order to more clearly highlight outliers and discrepancies and areas of interest inviting further exploration.

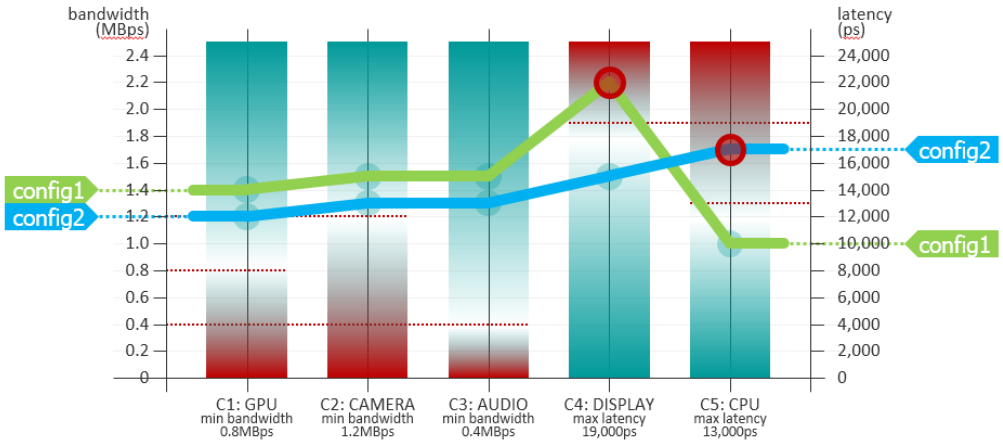

Figure 9. Overlaid data with mouse hover hot spots, variable user cursors and limits for contextual analysis

Customers and EDA companies alike can use the D3.JS library to implement custom visualizations for their own application domain, for statistical analysis, or debug of activity across a design, or correlation and analysis of detailed design behavior in a specialized area.

Figure 10. Examples of custom build visualizations for design data: statistical, traffic, activity, listings

VII. Other Areas of Focus

Alongside slick presentation of data in a form that invites interaction and discovery, there are other aspects of data visualization for SoC design/verification debug and analysis flows which are worthy of discussion. One of these is the use of surplus CPU power to the user's benefit.

A typical graphical debug tool with user interaction displays static information with each mouse click or key press or menu click, then leaving the user to digest the information or scroll through it before considering their next move. Sometimes the initial rendering time for each 'step' is significant enough to break the user's train of thought or concentration. There is an opportunity there. The fact is, during those 'idle' seconds of tool operation awaiting the user, there is no other activity from the software and CPU other than to do everything possible to aid the productivity of the expert operator of the tool. That means that there is nothing the tool should not do because it takes more processor cycles - it should do everything in its power to aid that productivity.

This kind of optimization can include GUI presentation layer improvements such as placing windows in the right place, animating them, moving them to the side while new information pops up and is populated. Also, any preprocessing or pre-analysis of data that may be revealed in the next scroll or navigation change.

In general, going all out to guide the user towards the information he seeks, iteratively, repetitively, without getting in his way or causing him to pause 'to operate the tool' even for a fraction of a second. Every millisecond counts and every millisecond should be used to benefit the user and help the most important analysis and visualization tool – their brain – keep its powerful processing pipeline full and productive.

Acknowledgment

Thanks are due to my colleagues in Mentor's Verification Methodology team.

.

References

[1] P. Russom, "Big data analytics," The Data Warehousing Institute, TDWI best practices report, Fourth quarter, 2011.

[2] L. Rizzatti, "Point/Counterpoint: Hardware Emulation's Versatility", http://electronicdesign.com, Dec 2014

[3] M. Bostock, "The D3 Gallery", https://github.com/mbostock/d3/wiki/Gallery, Dec 2014

[4] M. Bostock, "D3.js Blocks",http://bl.ocks.org/mbostock