Tough Verification Challenges: Data Visualization to the Rescue

Shaji Kunjumohamed
Broadcom Corporation
email:shajikk@broadcom.com

***Abstract*—**Designing and verifying an SoC is challenging. Design complexity is exploding at the same time engineers have to deal with reduced time for development due to ever-shrinking schedules, while at the same time not compromising design quality. Design teams have to collect and analyze large amounts of data produced by electronic design automation (EDA) tools as part of the sign-off process. Most of data analytics is manual, repetitive, error-prone, and can be difficult to automate. EDA data mining and extracting useful results in a timely fashion is the key ingredient to successfully taping out a chip. There are discussions about the application of "Big Data" analytics in the EDA space but a well-defined methodology to analyze EDA data using them is yet to be identified. An important piece of data analytics is data visualization and identification of points of interest. This paper proposes a method of converting repetitive, critical tasks like analyzing EDA data to something that developers can easily visualize. Manual tasks can be broken down and spread across the chip-development cycle.

I. INTRODUCTION

Recently, the growing amount of data generated from various data sources, such as consumer information-sensing devices, software logs and the like, have led to the development of advancements in Big Data analytics and related tools. Many methods and utilities have evolved that analyze and extract useful information out of huge amounts of data

An important part of Big Data analytics is data visualization. Humans can easily identify causality and trends in data if the presentation of such data lends itself to visualization. A good example of data visualization is the waveform viewer. A simulator running an RTL/gate simulation can be considered a data generation machine. The data is selectively sampled and routed into the waveform viewer using special tool constructs. Once visualized, the data makes perfect sense to a trained eye.

Data generated in EDA space has the same characteristics as that of Big Data. Huge amount of data is generated by various EDA design tools, simulations, scripts etc. A variety of data is generated; it can be either structured or unstructured. Data can come from multiple sources. As design complexity explodes, the data generated by various tools that analyzes same also increases at a greater rate.

In the hardware design space, as time progresses we see the evolution of new design standards and methodologies. All these ultimately translate to different tools from a variety of vendors. Diverse design teams may work using a combination of these tools to achieve their design objectives. A byproduct of this activity can be large amounts of data, represented in a variety of formats.

II. CHALLENGES

Time-to-market requirements, shortened spin-off times, and SoC design complexity pose many challenges today. To assure the quality of the design, designers are required to run a variety of tools and analyze the results. Data can flow from one design stage to another design stage or from one tool chain to another. Most of the time, the tool may be wrapped in an in-house flow that can modulate the data dumped out by the tool.

One of the challenges faced in the chip-development space is rapid identification of issues and fixing the design accordingly. The cost of development increases if the issues are identified later in the design cycle. If the issue cannot be fixed, the result can be catastrophic.

By nature, most of the design processes are iterative and manual. Many EDA tools are run multiple times throughout the design process and engineers are required to review the reports over and over to make sure the identified issues are fixed and no new ones are generated. Some examples are:

a. The running and analysis of tool reports, such as code coverage and functional coverage. Many times the process is repeated until all holes have been accounted for.

b. Reviewing regression results: regressions are run multiple times and holes fixed until some convergence is observed.

There are few patterns that here. The first is "human," the second is "manual" and the last one is "repetitive." When a task is manual and repetitive, humans tend to overlook obvious problems. This means some kind of automation should be developed to minimize the impact of this problem in chip development.

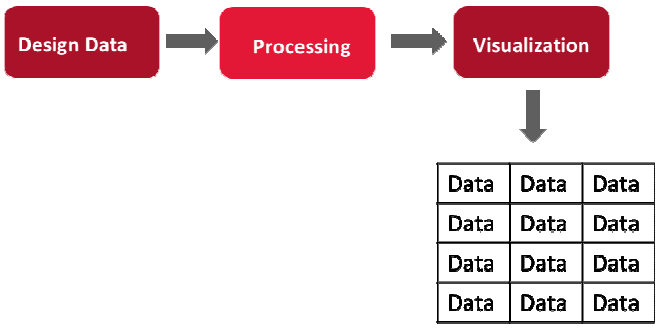
The developmental activity performed on a chip is not uniform throughout the development cycle. Towards tape-out, there is usually a spike—a "mad rush" to get all reports clean and hence most of the designers and verification team members are multitasking. Consequently, the development effort required by team members is not uniform over the development schedule and most of the important items are accomplished just before tape-out. Stress level increases and as a result, the risk involved also increases. Even though there are proper procedures set up, designers can miss important aspects.

III. Data Visualization Example

As mentioned earlier, data visualization is a generic term that denotes effort to help people understand significance of data by placing it in a visual context. We already use data visualization—it can be as simple as a table that has rows and columns that represent a general relationship between various elements, or it can be complex, such as a schematic viewer. It can have interactive capabilities, like the sorting of a table or putting markers/zoom on a visualization.

Another good example of a data visualization tool is the "diff" or "tkdiff" program that we are all familiar with. Just by using these utilities, we can visualize the difference between two text files.

EDA data dumped by different tools can be structured, unstructured, or semi-structured. Waveform files, sdf files, gate netlists, and tool logs are some examples. Data from these sources can be filtered and cleansed using scripts and relevant information can be extracted. It can be put in the form of a table to create a simple visualization (example 1). The only requirement here is somebody should know what data needs to be extracted and the format of the original data source. We will stick with "table" visualization for the rest of our discussions, since that is the most common form of visualization people use. This can be extended to other forms of visualizations as well.



Example 1

Let's look into an existing method of inspecting design data dumped out by tools. In general, the tools read in the design in a Linux environment and dump out reports in a text format. Some of the tools may have their own graphical user interface. Advanced users prefer to run the tool in batch mode and examine the text/log reports using a variety of tools, such as a text editor or text grep-ing/filtering using scripts. This may pose problems for those users that do not have access to such systems/software but may want to examine the data. To solve the problem, the data that needs to be examined can be also made available through a web browser.

Nowadays, the web browser has become one of the most widespread platforms for information exchange and retrieval. Browsers are available in all compute platforms and have the ability to support complex visualization. The simple table with EDA data discussed earlier can be easily rendered in a web browser using minimal code.

The volume of design data displayed in the browser can pose many challenges. It may not be practical to display full data in the browser due to performance considerations. In case of a table, only the data that fits into the user's view port need to be rendered. The user should have the ability to sift through the data presented in the table by means of sorting the table or by using filter functions in the table. Each time users apply a filter or sort function, data can be streamed into the browser from an intermediate external source, such as a database. This enables the user to filter and view only the interesting data.

IV. Recording User Inputs

So far we have described a basic framework for displaying data in a browser. When we present data to users for review, they may go through data and figure out interesting observations. For example, if it is chip-level connectivity data that users are looking into, somebody may observe a bad connection. The discrepancy should be recorded and the design fixed. The table visualization can be easily modified to attach user comments and status with each entry. A variety of widgets such as checkboxes, textboxes, and in-place editing toolsets are available to achieve this. Once saved, the user inputs get attached to data and get saved back into database. When somebody else opens the table from a browser from a different device, he or she may have full access to view the data together with the user inputs. It may be possible to filter data based on user comments—for example, somebody can sort data based on bad connections and immediately start fixing the original source code.

The final missing piece in the above system is the addition of accountability. When a specific user adds a comment or a status, it can be saved with the user ID and timestamp. In case of design flaws, it can later be traced back to the reviewer. If an authentication mechanism is implemented, user names can be appended to status and comments along with timestamp. At this point, the system may truly reflect a collaborative environment.

Sometimes the tool environment may need access to user inputs for refining the next tool iteration. A typical example of this is where a user waives certain functional coverage points by clicking the "waive"' checkbox. The underlying tool may want to access this for the next run. Once the user input is saved into the database, the tool can access the checkbox value from the UNIX/Linux environment and can use the same to configure itself.

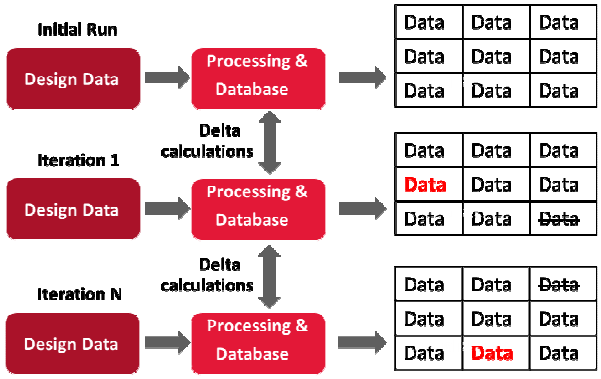
V. Chasing a Dynamic Design

The development schedule of a typical chip design project runs for months until the project is wrapped up. During this period, the design is very fluid and changes as new IPs are integrated. New connections can appear and existing ones can become obsolete. Designers run various tools periodically to quality-check the design. Since the design constantly changes, the data and reports generated by the tools also change. Care needs to be taken to ensure that no existing features are broken while newer ones are added as per the specification. This is easier said than done—it requires that team members constantly monitor tool reports. One way to tackle this is to keep reviewing data until the design is frozen. The designer may end up spending energy reviewing items that have already been reviewed multiple times and can easily miss design flaws that appear in reports. Since everything is dynamic, engineers may choose to run certain tools towards the end of the design cycle when the design is stable, closer to tape out. This indeed is a very common practice across many design teams. There are risks involved here, since there can be a spike in effort involved. Individuals may start prioritizing tasks that are deemed important and may choose to skip finer inspection of data.

One way to solve the above problem is by evenly distributing the effort involved in data inspection throughout the design cycle instead of towards tape out. This can be achieved by data processing/visualization techniques. The raw data that comes out of a tool can be processed to extract only relevant information. The intermediate data can be stored into a database. For the second iteration, data processing techniques can be employed to find out the delta between the previous iteration and the current run. There can be differences and these can be presented in visualization by appropriately flagging the entries. The user can then focus more on changes in the design since the last tool run. When a delta is detected and resolved, comments and status can be added to it and it can be saved back into database along with the complete history information.

The system can be designed to run at predefined intervals as soon as the project takes off. Users can spread the work evenly throughout the chip development cycle rather than increasing it at tape-out. The effort level comes down drastically and risk to the project is minimized.

Example 2 represents the idea in brief. In the initial iteration, the data is extracted and processed. After data cleansing operations, it is saved into a database. The visualization tool (browser) fetches data from the database based on the user requirements. The user analyzes data, adds comments and status, and saves data. The data gets saved into the database and will be available to all users of the system. During the next iteration, the process is repeated. Deltas between each run are presented to the user. In the given example, the data in red refers to new/modified data and the data in strikethrough format represents deleted data. When a new visualization is generated, the comments/status from the older iterations can be rolled over or imported to the new visualization. This way, users may not waste time in reviewing data that has been reviewed previously. They can focus only on the changes flagged by the platform.



Example 2

VI. Other Considerations

EDA vendors generally provide a tool or utility to analyze the design and generate reports. It is up to the user to decide how to interpret the tool output data. The size and architecture of designs may vary between design teams. Depending on their requirements, designers may choose to use a specific vendor tool to achieve a design goal or they may use an assorted list of tools. Hence the data obtained from flows using these tools can be very diverse. Again, each team may choose to focus on different aspects of tool data—for example one group may be focusing on achieving full code coverage on their design. Others may be more interested in achieving more functional coverage. In summary, it is not possible to create a generic platform to address everybody's needs. In addition, each team may have individual preferences on how the data needs to be presented. For example, hardware design is organized hierarchically. One team's requirement may be to capture data from all hierarchies in a single web page, yet others may want data to be more organized—per block level.

Now the question remains whether it makes economic sense to invest in developing such a web application. The following are the obvious benefits:

1. Chip data is collected, organized, and documented. Every aspect of the review process is recorded.
2. It cuts down development time and brings down the effort involved.
3. It reduces risk.
4. Once historic data is available in an organized form, users can do interesting data analytics, for example, comparing different versions of designs.
5. The system can be reused or shared between different groups in the company.
6. The system can be built ground-up from freely available open source tools, so licensing fees can be minimal.

Obviously there are technological challenges in implementing such a system, but the end product will be worth that investment.

VII. Technology Involved

The following components are needed to implement the proposed platform:

1. Script infrastructure to filter needed data from EDA log files or tool data. Data cleansing and extraction operations can be done here. Infrastructure based on languages such as perl, python, etc. can be used. The developer should be aware of what data is relevant and what needs to be extracted.
2. A database to store the data. A data base is a program that runs perpetually on a machine that has the capability to store and retrieve data. Various standard database technologies are available, most notably NoSQL data bases like MongoDB.
3. A web browser on the user's device. The web browser connects the user to the web server. The visualization application will be physically located in the webserver and it will be downloaded to the browser from the webserver for execution. The applications executed in a browser are typically written in JavaScript and HTML (the front-end, user-facing part of the visualization application). Many open source front-end frameworks like Angular.js, D3.js and Bootstrap are available that further create higher-level abstractions of the front-end development tasks. All the code that is written for a browser is called "client side" code. The data needed by the application that runs on browser comes from the data base.
4. A web server that hosts the application code. The web server helps to connect a browser with the database after authenticating the user. Many web server implementations are available, such as Apache, which is difficult to set up and configure. Now there are newer technologies like Node.js that developers are adopting, which can be easier to configure and use. Frameworks developed on top of Node.js, like "Express.js," make configuration and use of the web server very easy. All the code that is written for the webserver is called "server side" code.

A variety of open source software bundles that integrate the technologies described above in a concise fashion exist today. One such bundle is called a MEAN stack (meanjs.org). It makes use of MongoDB, Express.js, Angular.js and Node.js. The beauty of this framework is that all the components can be configured using only one programming language: JavaScript, as opposed to other technologies that involve a variety of programming languages specific to tools. This makes it very easy to develop and deploy applications using MEAN.

VIII. Implementation Aspects

The MEAN framework can be checked out of GitHub, as described in the MEAN website. The dependencies (MongoDB and Node.js) also need to be installed in the target machine. Once the MEAN framework is installed and configured, it can be a matter of minutes before the first sample site can be ready.

The developer can also set up an authentication system that comes with the MEAN framework. It can be easily integrated with the company's LDAP authentication scheme. Depending on the design data, the developer needs to come up with a schema. This is a simple JSON data structure that describes how data will be stored in a MongoDB database. The schema should be set up in such a way that it also captures the time stamp and user ID of the end user. Additional fields also can be added to distinguish tool iterations.

Once the data cleansing/extraction scripts that parse the EDA tool data extract the relevant information, the MongoDB database can be populated from the extraction script itself. The population of the database needs to use the same schema described above. Once populated, the visualization application running in the browser will have access to the data.

Another important aspect of the user interface (UI) is the aesthetics of the visualization. The Bootstrap JavaScript library can be used to improve the look and feel of the site. For example, for the table visualization the end user may request to organize a table with a variety of UI features such as paging, filtering, accordion, and the like. All these can be integrated into the web page with minimal code.

The various components of the MEAN framework are widely used in web development. All the tools used in the framework are open source and heavily documented. There are tons of resources and example code available on the Internet that can help anybody with the development of complex applications.

IX. Practical Examples

The system described above can also be extended to other areas of chip design, not just front-end design and verification.

Many design teams use checklists for signoff purposes. It is generally a shared spreadsheet, checked into version control. As the spreadsheet grows in size and complexity, it becomes a problem to maintain. The above system can be used as an alternative to a spreadsheet. It can automatically store a history of edits, names of collaborators, etc. Specific UI components can be added to make this a living document.

For example, if a specific checkbox is selected in the webpage, the value of the same (checked/unchecked) can be assessed from the UNIX/Linux side. Tool configurations or wrappers can have access to the same value.

Another interesting application can be online test plan generation and maintenance. Specific widgets are available that support in-place editing of text in a webpage. Users can check/uncheck various scenarios in a test plan that are relevant to the product. Regression results can be automatically parsed and imported into the test plan. This will give an accurate idea of how much functionality in the design has been tested and how much is passing or failing. Most importantly, users can

compare the previous regression runs with the current regression run and analyze what exactly changed.

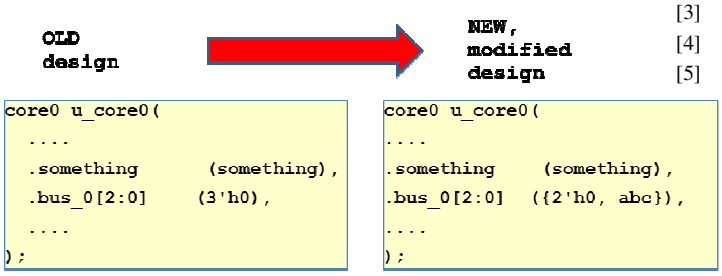
Code reviews are an important part of the design process. Most design teams review the connectivity at the top level of the chip with various IP owners. The code to be reviewed may amount to hundreds of thousands of lines. Many interesting observations and bugs can come out during code/connectivity reviews. It is a highly collaborative process. Formal/assertion tool-based reverse connectivity can be used to extract the connectivity of the top level. This can be converted into a connectivity table for users to review. They can go through the data and put in comments and status for each entry. As time proceeds, the design may change. After each design iteration, the data can be extracted and presented in such a way as to allow easy visualization of the changes, much like "tkdiff" works for basic text. The users can review only the changed information and sign off. This process can repeat until the design stabilizes and the project is taped out. This system can potentially save hundreds of man-hours and can also create good documentation of the design connectivity.

To illustrate this further, the connectivity of a block in the top level of a chip can be represented in a table as given in Example 3.

Block	Direction	IO Name	Connected To	Latency	Comment	Status
u_core0	Output	o_ack[2:0]	xyz_inst.i_ack[2:0]	2	Good	☑
u_core0	Inout	pad_a1	abc_inst.i_pad	-	Wrong	☐
u_core0	Input	i_rst[2:1]	2'b01	-	Looks good	☑
u_core0	input	i_rst[0]	unconnected	-	Check with designer	☐

Example 3

Designers can browse through the table and update the status and comments. During review, they can flag erroneous connections. As time proceeds, the design may change. Example 4 shows "old design" and "new modified design" with a few changes in connections. Connectivity can again be extracted and presented in a similar table as given in Example 3. Since we already have the connectivity data for the old design, data can be processed so that the comments and status of those fields for which connections has not changed can be rolled over to the new table.



Example 4

Block	Direction	IO Name	Connected To	Latency	Comment	Status
~~u_core0~~	~~Output~~	~~bus_0[0]~~	~~1'b0~~	-	~~Good~~	☑
u_core0	Output	bus_0[0]	abc	-		☐
u_core0	Output	bus_0[1]	1'b0	-	Good	☑
u_core0	Output	bus_0[2]	1'b0	-	Good	☑

Example 5

Example 5 shows the case where some connections are changed. In the "old design", all bits of bus_0[2:0] were tried off to zero whereas in the "new modified design", bit bus[0] is connected differently. In the table visualization, such changes can be highlighted as given in Example 5 and the reviewer can easily spot such delta changes in design. For example, the reviewer can sort and filter the table only to view unchecked checkboxes and it represents the new connections in the design, not yet reviewed. Hence this type of visualization aids team members to review only the changes in design as opposed to reviewing complete dataset, especially when there are thousands of connections to be reviewed.

X. CONCLUSION

For a given EDA tool report/data generated on a design, knowledge about the data points of interest and feedback on how data needs to be organized visually only comes through user experience. The information is not generally translated to specific utilities that industry can develop and use. This paper describes a way in which semiconductor companies can rapidly develop and prototype data mining tools in-house using an inexpensive MEAN development stack that combines the power of software tools like MongoDB, Express, AngularJS, and NodeJS. While there is initial effort involved in implementing such a system, the benefits of having it far outweigh the initial investment. Examples from a real-world scenario involving code coverage and formal connectivity were presented.

XI. REFERENCES

[1] http://meanjs.org/

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

[3] https://en.wikipedia.org/wiki/D3.js

[4] https://en.wikipedia.org/wiki/Node.js

[5] https://en.wikipedia.org/wiki/MongoDB