2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

Architecturally Scalable Testbench For Complex SoC

Senthilnath Subbarayan, Senior Staff Engineer, Qualcomm, Bangalore, India
(subbaray@qti.qualcomm.com)

Arulanandan Jacob, Staff Engineer, Qualcomm, Bangalore, India
(jarulana@qti.qualcomm.com)

Sandeep Kumar, Senior Staff Engineer/Manager, Qualcomm, Bangalore, India
(ksandeep@qti.qualcomm.com)

***Abstract*— SOC design complexity increases by multifold year on year. To address the growing complexity of design, it mandates the need of optimized testbench to adhere strict time to market scenarios. This paper talks about the art of constructing SOC testbench, to verify any data path by bringing in RTL modules and the required testbench components by configuration approach. The configuration file used, takes leverage of the same inputs used by the Portable Test and Stimulus Standard (PSS) for creating testcases. Paper also discuss about the ease of use of the testbench along with time saving and memory footprint obtained through this configurable testbench approach.**

Keywords*— *scalable testbench; black boxing; PSS; UVM; performance

I. INTRODUCTION

In today's fastmoving consumer electronics industry, achieving short time-to-market is the single most effective way of maximizing semiconductor sales and profit. As the size and complexity of SoC design grow each year, there arises a need for optimized design and verification integration techniques to cater the needs of a quality product.

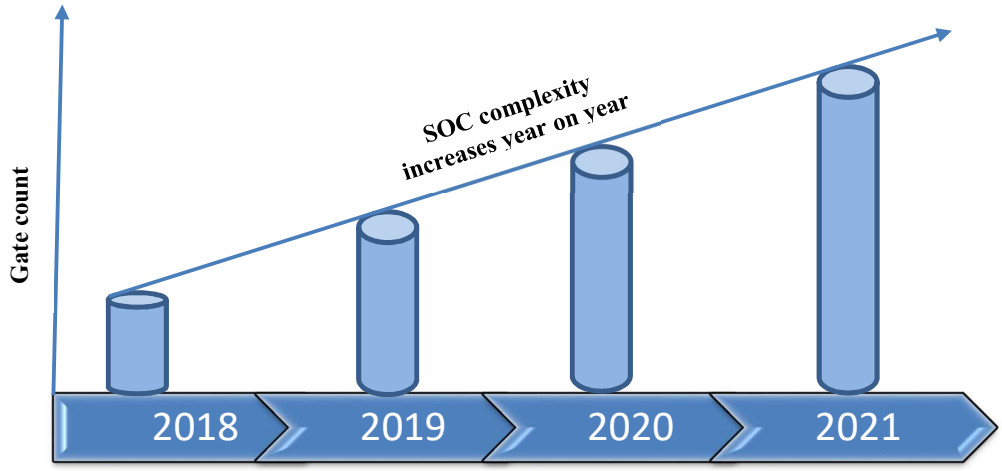

Figure 1 Gate count vs time

Though there are various automation techniques available starting from Register Abstraction Layer Modelling (RAL), testbench build with integration of Bus VIP's, to speed up IP verification, there exist a limited option to speed up testbench bring up time for SoC testbenches. IP-XACT representation of IP and verification IP (VIP) provide the standard way of exchanging SoC design information, it takes a significant time and effort to modify the in-house IP's and VIP's in IP-XACT format and then create a SoC testbench with self-developed utilities or vendor tools.

In SoC verification, the arrival of IP's at different times for SoC integration mandates the need for an adaptable testbench to meet the tape-out deadlines.

In this paper, we will be discussing about the need for scalable testbench and existing testbench solutions. Later part discusses about how Architecturally Scalable testbench provides a solution for easy integration of testbench components along with dynamic RTL black boxing mechanism to get the best build and run time.

II. NEED FOR SCALABILITY

When the product portfolio increases, a generic testbench to cater all the diverse product needs is limited. Maintaining such a generic testbench becomes bulkier over a period. To use the generic testbench for a specific product, there arises a need to bring in the required verification components, integrate and verify this product. Integration of such verification components manually, is time consuming and may lead to human errors.

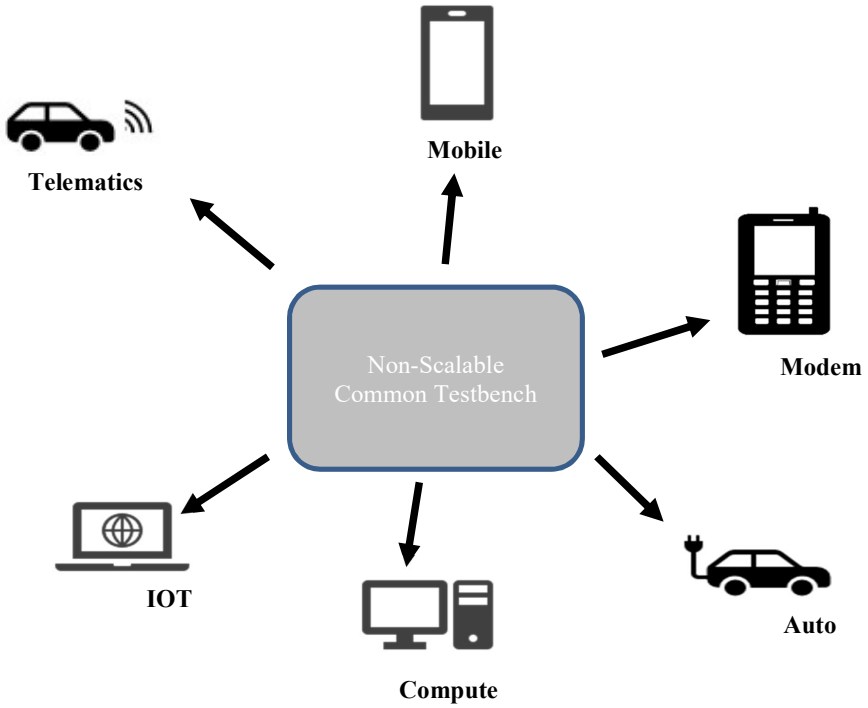

Figure 2 Generic Testbench for Diverse products

There arises a need for testbench components integration in a modular fashion to reduce complexity/compute time with increased efficiency.

A. RTL Less Compile

To adhere strict SoC timelines, there is a need for RTL less testbench compile. Before receiving the first level integrated RTL, the SoC testbench should be integrated with the required verification IP's (VIP) based on the I/O interfaces. This enables the scope to automate the VIP integration and the number of VIP instances in the testbench instead of manual integration. Here, it enables a need for smart integration of VIP's and its instances by automation.

B. Compile time switch Reduction

When a SoC chip is a derivative of previous chip, the testbench and its components are reused. Any additional/unused components are added/removed from the SoC testbench by means of compile time switches. Re-use being the most convenient way, any manual Testbench component modification will result in significant increase in time to stabilize the Testbench. This also increases the Testbench size and number of compile/run time switches. Maintaining the Testbench over a period of years become cumbersome. Here, it enables a need for smart integration mechanism to add or remove testbench source codes for easy maintenance.

III. EXISTING TESTBENCH SOLUTIONS

Most of the tool vendors support IP-XACT format of data representation for their IP's and VIP's so that the interconnection of design and verification components are taken care by the tool and an automated testbench is created. This approach is good when all the underlying IP's and VIP's are IP-XACT representable, but in a large SoC's where the most IP's and VIP's used are internally developed, it requires a huge migration effort for IP-XACT representation and stabilizing the generated testbench. Also, vendor license is required to enable automation in testbench generation. Though the solutions are not readily available to be used, it allows us to explore for other solutions.

The Architecturally Scalable Testbench (ASTB) provides an automated way to integrate/remove any component into the testbench based on the configuration and migrate the same across different SoC platform.

IV. ARCHITECTURALLY SCALABLE TESTBENCH (ASTB)

The first step in developing a scalable testbench is to create a configuration file. A SoC UVM Testbench consists UVM constructs to instantiate, build and connect the required UVM components. The configuration file should contain entries that are required to construct an UVM based testbench. The constructs required for creating instance and building the testbench components are brought in through template file. SoC testbench can also consists of reference models that are required to verify a specific block. The source codes of such reference models are brought into the testbench through marker file mechanism with the help of a utility.

The user input configuration file is created in such a way that it can derive the required information from Portable Stimulus Standard (PSS) input file.

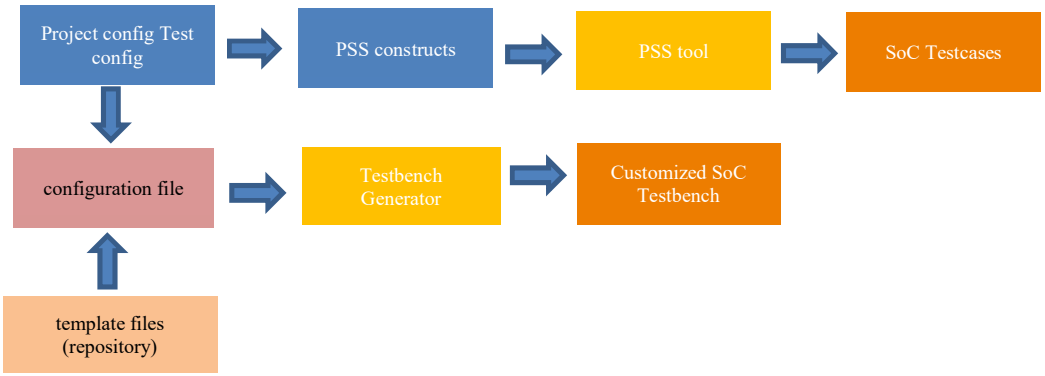

Figure 3 Config file generation from PSS flow

ASTB provides an easy and convenient way to integrate the testbench component, thus reducing the efforts on migrating the testbench across different platform. It also enables faster debug time due to black boxing of the unwanted RTL modules. Overall simulation time is reduced due to lesser components in testbench and RTL.

In ASTB flow, the blocks required for simulation is selected instead of using the full SOC Testbench. Similarly, the RTL modules which does not contribute to the data path are black boxed as per the configuration file. This results in usage of optimized testbench along with optimized RTL for a given simulation which reduces memory foot print and elaboration time. This also promotes the standardized way of integrating Reusable Testbench components for a given chip.

Below flow diagram [Figure 4, Figure 5] compares between the conventional testbench and ASTB.

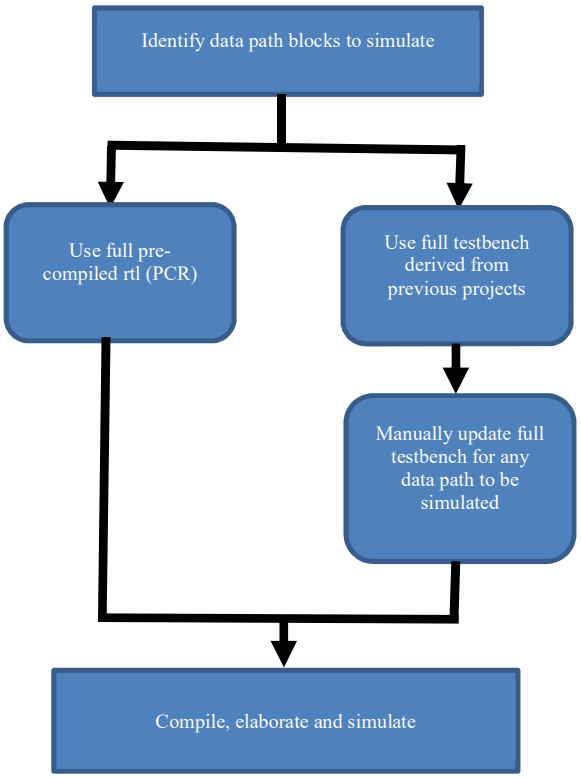

Figure 4 Conventional SoC Testbench Flow

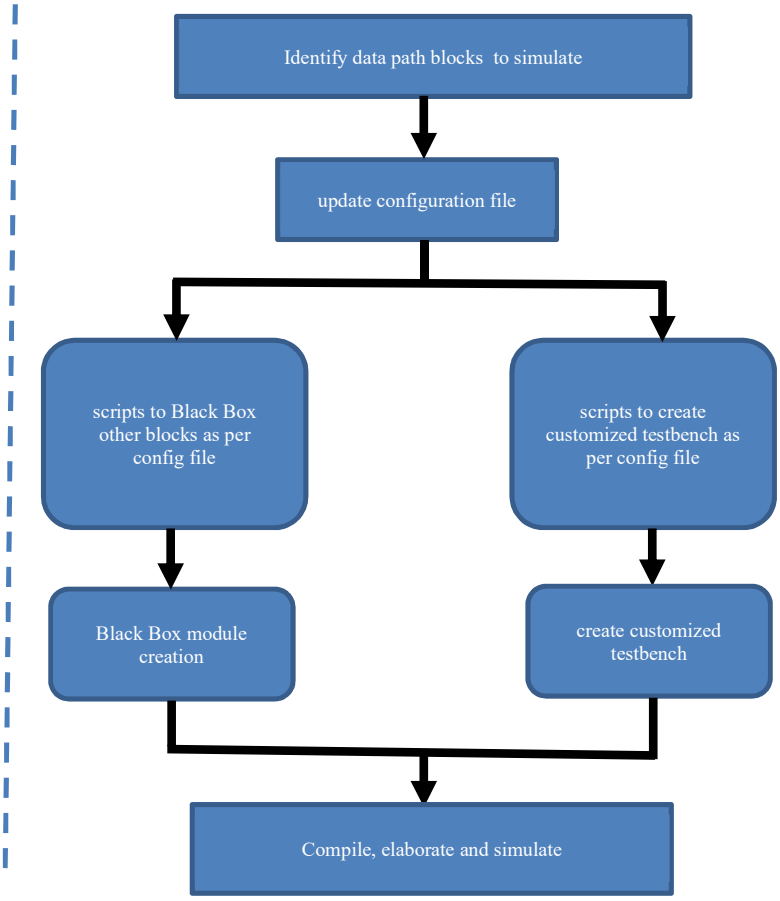

Figure 5 Architecturally Scalable Testbench Flow

V. RESULTS

We have chosen the following data paths for comparison. It comprises of external debug interface, test controller interface, Processor 1 and Processor 2. The build and run times are compared with conventional testbench and ASTB.

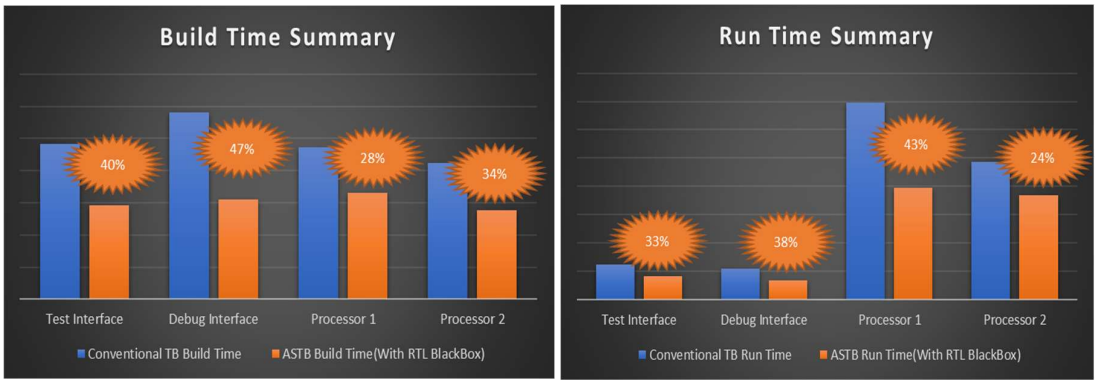

Figure 6 Performance of ASTB with conventional Testbench

The memory utilized by ASTB over conventonal testbench is as follows. Here, we see around 35% reduced memory usage over conventional testbench.

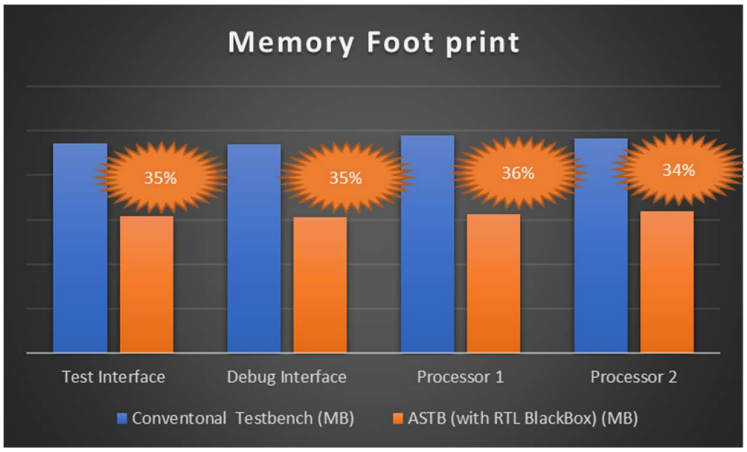

Figure 7 Memory usage of ASTB with conventional Testbench

VI. LIMITATIONS

ASTB supports for RTL simulations in SoC. This can be extended to Emulation, Vector, Gate Level Simulations at SoC level. Currently, SoC level Testbenches are supported by ASTB. This concept can be enhanced to support IP level environment as well so that it forms a single Testbench that can be used by different stakeholders.

VII. CONCLUSION

This paper discusses about the approach that can be used to derive a platform independent testbenches. It suggests a practical application method to build an efficient and structured verification environment which meets various requirements of SoC verification. The scale of SoC complexity varies with respect to product, may it be a simple IOT device or a complex mobile phone, we need a mechanism to generate platform independent testbench in less time. ASTB address this problem, along with dynamic RTL black boxing reduces the build and run time around 30 to 40 %. The surplus time can be used to verify additional system level scenarios.

REFERENCES

[1] UVM based testbench architecture for logic sub-system verification T M Pavithran ; Ramesh Bhakthavatchalu 2017 International Conference on Technological Advancements in Power and Energy (TAP Energy)

[2] https://www.testandverification.com/wp-content/uploads/DVClub/19_Mar_2013/Broadcom-Lavanya&Balaji.pdf

[3] https://www.design-reuse.com/articles/18478/semi-automatic-testbench-generation.html

[4] https://link.springer.com/chapter/10.1007/978-3-642-42024-5_34