2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE
MUNICH, GERMANY
OCTOBER 15-16, 2024

Functional Verification Using C Model: DPI-C VS Static Value Tables

Djordje Velickovic, Verification Engineer, Veriest Solutions, Nis, Serbia (djordjev@veriests.com)

Katarina Bozinovic, Verification Engineer, Veriest Solutions, Nis, Serbia (katarinab@veriests.com)

***Abstract*— This paper describes two popular approaches for usage of a code written in C language in functional verification process. These approaches are integration of C model using DPI-C System Verilog function and usage of static and pre-generated in/out C model value tables. The paper gives a short overview of their implementations and compares their performance through a case study set on an exemplary design. It also gives comparative analysis between these two approaches in terms of effort, range of suitability for different design cases, reusability and integrability. The goal of the paper is to help and guide verification engineers in choosing the most appropriate method of C model integration into their verification environment, for their specific needs.**

Keywords— System Verilog, UVM, Functional Verification, DPI-C, C model

I. INTRODUCTION

Use of C modeling in functional verification is pretty widespread in today's industry. Traditionally associated with DSP related design, C models in design and verification processes are being used seemingly more than ever. The need for efficient and standardized use of C models during verification comes with ever increasing complexity of the mathematical manipulation of the signals inside such design modules. This is commonly done for complicated designs, when block contains mathematical calculations and statistics, that are hard to mimic in a verification environment as a reference model. The general principle is to have a dedicated team who will create a model in the form of C language function, which will produce the results of mathematical manipulation of the signals as described in the specification of the module. Then it's up to a verification engineer to integrate the C language model and verify the design.

When it comes to the process of using a C model for verification purposes there are multiple solutions circulating around the industry. Two of the more popular approaches for C model utilization in functional verification are:

- Integration of C model using DPI-C System Verilog function
- Usage of static, pre-generated in/out C model value tables

Both of these methodologies are well established within the industry and have been in use for many years. Over the years, they have been refined and they received multiple variations on implementation guidelines, driven by various specific project needs. When two solutions are established for a unique problem, in this case problem of C model utilization in a verification process, there is always a question, which one to use. This paper will try to give an answer to that question by comparing two approaches mentioned above. It will do so, by giving brief overlook on implementations of integration of C model using DPI-C function and usage of pre-generated C model value tables and showcase results from simulations on a selected case study, providing a quantitative basis for comparing their effectiveness. For the purposes of the study, UVM based environment [1] is built with an option to use either of the two approaches and therefore provides a way of comparing their performance in multiple categories. The goal that this paper aims to achieve is to give a guide for verification engineers in choosing the most appropriate method for their specific needs, ultimately enhancing the reliability and efficiency of their verification processes when verifying DUTs which are mandating usage of a C model.

II. IMPLEMENATTION

A. *Integartion of C model using DPI-C System Verilog function*

Integration of C model using DPI-C [2] system Verilog function relies on a DPI (Direct Programming Interface) feature of the System Verilog language. DPI is an interface between System Verilog and a foreign programming language [3]. Further, DPI-C allows direct inter-language function calls between System Verilog and any foreign programming language with a C function call protocol and linking model. This feature gives the ability to verification engineer to call a C function which contains mathematical calculations generated by an algorithm specialist anywhere from regular verification checking flow.

One of the most common ways of integrating the C model using DPI-C System Verilog function is through a scoreboard component of the environment. Monitors of both input and output interfaces are connected to the scoreboard through ports as usual in UVM based verification environments. When a transaction from an input interface monitor arrives in the scoreboard, it is unpacked and a value which represents the input signal, is being used to calculate an expected value of the signal at the output interface. This is being done by calling a DPI-C function with a value, unpacked from an item, received from the input interface monitor, as an argument. When the monitor of the output interface sends a transaction to the scoreboard, the value representing an output signal is being compared to the predicted value, which was calculated using the DPI-C function. In this way for each valid transfer through the module, the verification environment confirms if the design is outputting a correct value for any input value that has been driven at its input. A diagram showing verification environment which integrates C model through DPI-C function and uses it for data checking is presented in Figure 1.

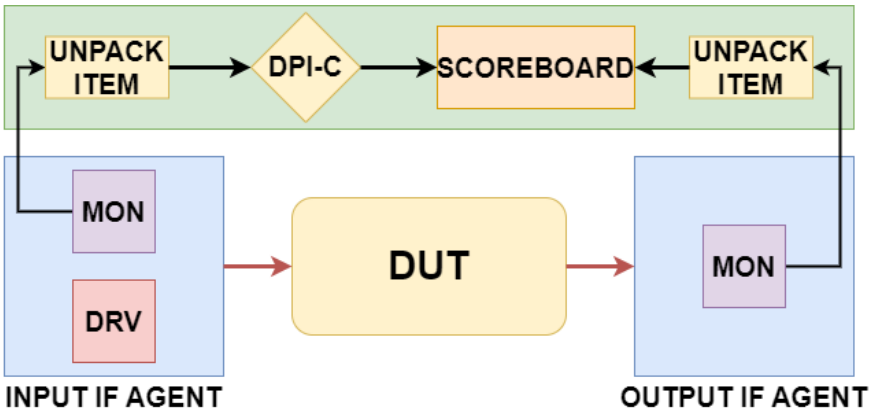

Figure 1 - Verification environment diagram with integrated DPI-C function

As said earlier, a C function can be called anywhere from the checking flow of the environment which means that implementation through scoreboard is not mandatory.

B. *Usage of static, pre-generated in/out C model value tables*

Static, pre-generated input/output values can be provided by a design-based model written in C, C++, HDL or MATLAB code [4]. The verification team can then use simulation-based testing and static analysis to complement the model-based design to find errors faster and in the earlier stages of testing [4]. The team which handles the C code model needs to generate a table of input values and the corresponding results table presenting expected values at the module output. This would mean that the process of creating reference input/output tables needs to be done in a separate flow before any running of the simulation. Input and output values are usually provided as a text file, that is further used in verification environment.

To create the stimuli, a verification engineer should implement a logic for loading values from the input table into the driver component of the agent connected to the input interface of the module. This logic should usually contain mechanisms for opening the input file, reading, tracking a position in the file, and flags for empty file or for end of file. In some cases, there is an additional file with instructions on how to read the input values table.

This file can include information on how data is ordered in the input value table, if there are several samples, position of the LSB and MSB bit, also information if there are control signals. Under some methodologies text files/tables are also called “traces”.

There are several ways how reading of the input trace can be implemented. One of them is to set the number transactions from a controlling sequence to be 1. This means that once the file is opened and the driver starts reading it, control will be returned to the test flow from the driver once the reading is complete. That means that the transaction will be fully executed once the driver is done with reading the input trace and driving read values to the interface. The second option, for example, is to treat each line in the trace as one transaction. The point is that since there is an input that is not completely controlled by a verification engineer then also the control of the test flow is different.

Commonly, In UVM based verification environments data integrity checking is performed in the scoreboard component of an environment. Data is first collected from interface by the monitor component and then sent as a transaction to the scoreboard. When using a static predefined reference, the output data checking is similar, with some exceptions. The difference is that in the scoreboard there is no logic for the reference model that provides the expected data. Instead of that, expected data is provided within a trace file, that should be read and used as a reference. According to that, logic in the scoreboard should contain code for repacking the data from the monitor transaction, providing expected data from the file, and comparison logic. Similarly to the input, for the output checking it is important to have instructions on how to properly read the trace file.

Some approaches can use testbench wrappers for checking the data, instead of the scoreboard component since there is no reference model logic. In that case, testbench should contain the logic for reading the file and all needed checkers on data output.

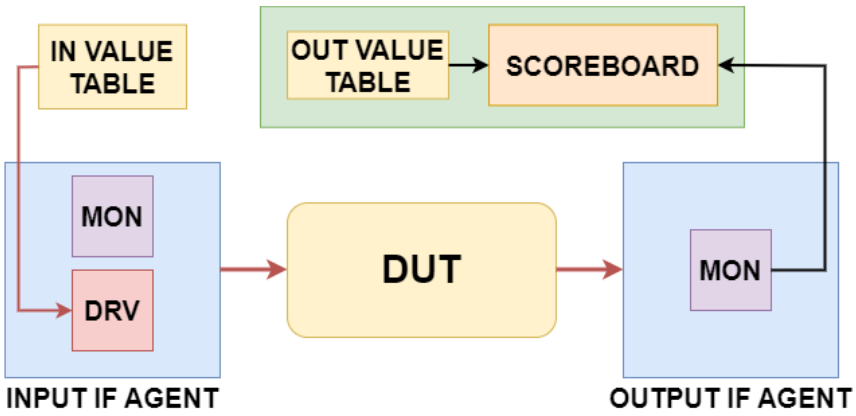

Figure 2 - Diagram of verification environment which uses pre-generated value tables

III. CASE STADY

For the purposes of this paper, a case study is set up with the aim to showcase performance differences of the two approaches of C model utilization in verification, explained above. Under the study, the verification environment is implemented with two variations – one that integrates C model through DPI-C function and the other which uses pre-generated in/out C model value tables. The verification environment is built around a simple CRC calculator module generated using open-source CRC generation tool [5]. Definition of the module interface is given in the table 1

Table 1 - CRC module interface signals

Siganl name	*Signal width[b]*	*Signal direction*	*Description*
crcIn	32	Input	Generator Polynomial CRC module input
data	32	Input	Data input of CRC module
crcOut	32	Output	CRC module output

CRC module used in this case study simply calculates CRC 32-bit function output based on a driven inputs of *data* and *crcIn*.

In order to establish the best possible conditions for getting most accurate performance measurements, development flow of the environments was slightly changed from usual. The method of generating stimuli differs between two approaches where DPI-C option relies on SV randomization on transaction level and static, pre-generated value tables are containing the stimuli got by executing the C code outside of the environment scope. To ensure that both environments are tested using the same data, pre-generated value tables are created by dumping data from input and output monitor components of the environment with integrated DPI-C function, into respective files, during the simulation performed with randomized stimuli.

IV. PERFORMANCE RESULTS

For testing of performances of two approaches, simulations were run on 1000, 10000 and 100000 back-to-back, data transactions. As mentioned, the same stimuli were used for both environments. During the simulation, CPU usage, memory usage and simulation time were measured. These simulation performance results are acquired using *-perfstat* feature of the Xcelium [6] simulator. All results presented as performance metrics are generated by averaging values measured over multiple runs of simulation.

A. *CPU usage*

Results from measuring CPU usage show that the implementation with pre-generated value tables is using less CPU resources than the implementation using DPI-C. Chart which displays comparative results of CPU usage is shown in Figure 3.

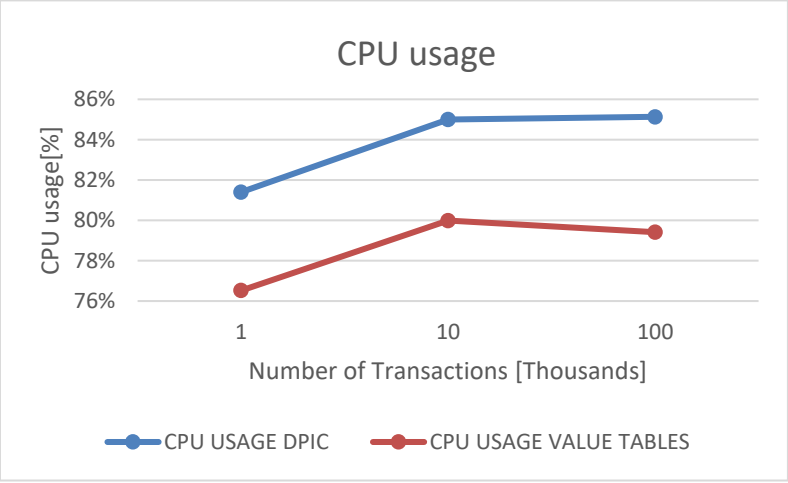

Figure 3 – Chart showcasing CPU usage results

This conclusion stays consistent with the increase in the number of transactions driven during simulation.

B. *Simulation time*

Simulation time results show that implementation with pre-generated value tables is more time efficient than implementation using DPI-C. Simulations running 1000 data transactions using pre-generated value tables are around 9% faster on average, while a 12% increase in simulation speed on average was observed running 100000 data transactions. Chart which displays comparative results of simulation time is shown in Figure 4.

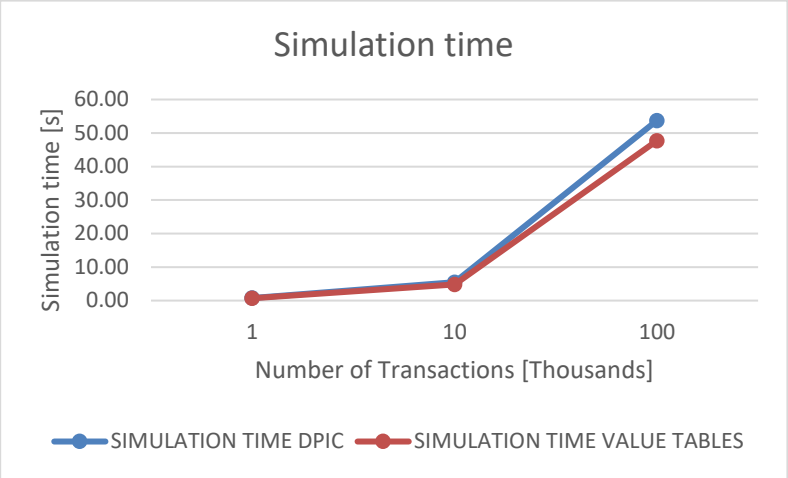

Figure 4 - Chart showcasing Simulation time results

Implementation using DPI-C function shows inferior performance in terms of simulation duration and this difference in performance could be significant. Assumption is that with increased complexity of design which is being verified and therefore increased complexity of C function, which is being integrated into the environment, difference in simulation time will only rise.

C. Memory usage

In terms of memory usage, results show that implementation using DPI-C is much more efficient. With the increase of transactions driven during the simulation, this implementation does not use more system memory. On the contrary, implementation using pre-generated value tables requires more system memory with an increase of driven transactions during simulation. It's important to say that the verification environment which uses value tables used in this experiment performs all data checks during check phase of the simulation and therefore needs to store all data collected during the simulation up to the end of simulation. In case there is a memory concern, this implementation can be optimized to use less memory by adding a mechanism to check data on the fly, after each driven transaction as described earlier in the paper. The chart which displays comparative results on memory usage is shown in Figure 5.

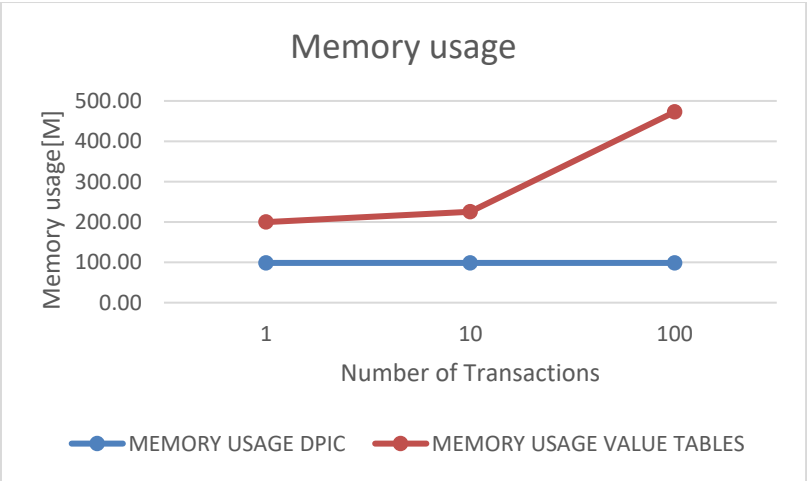

Figure 5 - Chart showcasing Memory usage results

D. Profile performce analysis

As part of performance analysis of two environments tested in this case study, profile analysis was conducted using a Xcelium profiling tool (*-profile*) which gave the insight in distribution of simulation time consumption in

TB and design [7]. Results of the profile analysis are given in table 2. Profile analysis is conducted on a simulation using 10000 transactions.

Table 2 – Profile analysis results

DPI-C		Value Tables	
TB/SV	65.8	TB/SV	79.3
Miscellaneous	18.3	Miscellaneous	17.9
RTL/Design	1.0	RTL/Design	2.3
Randomization	14.8	Randomization	0.5

Analysis shows implementation with DPI-C function spends around 15% of the simulation time on randomization while in implementation with pre-generated value tables this number is negligible. When these results are put into the context of comparative simulation time analysis from earlier, a conclusion can be drawn that creating an item on the fly and randomizing it with each data being driven into DUT significantly slows the simulation. If simulation time is of concern, and usually is, opting for implementation with pre-generated value tables instead of implementation using DPI-C can save around 10% in simulation time per findings of this case study and likely, significantly more for more complex designs.

V. DISCUSSION

A. *Integration*

When integration of C model using DPI-C System Verilog function and usage of static, pre-generated in/out C model value tables implementations are compared, the latter usually comes out as a simpler solution which uses less code. This simplicity and flexibility come from the fact that the verification engineer who is implementing a checking mechanism based on pre-generated value tables is basically reading from pre-delivered files to drive the stimuli and compare data. On some occasions, when value tables are not aligned to interface definition, data needs to be manipulated or repacked, but this is often true for the implementations using DPI-C function as well. Although solutions with DPI-C integration usually use a bit more code, DPI-C usage is well established in the industry and very well documented, with numerous tutorials and guides available on the web. Therefore, the skill threshold for mastering any of these two approaches is not high and both approaches are suitable to be used by less experienced verification engineers.

B. *Randomization and coverage collection*

Contrary to the clear performance advantages that pre-generated value tables solution is giving, usage of C model through DPI-C function has few practical advantages which could be very impactful. When the C model is integrated through DPI-C function, stimulus is created using constrained random features of the System Verilog language. [8] Verification engineers are generally accustomed to this approach and coverage is filled traditionally with running a regression of tests with randomized data. In order to reach full coverage using pre-generated C model results, verification engineer simply needs to run all configuration scenarios with any possible stimulus. For interfaces which are handling large data this can be quite difficult and lengthy. If the generation of the value tables is taking too much time or resources this can potentially nullify any performance gains in time simulation achieved by opting for this approach.

C. *Debug*

During the verification process, debugging of the design usually consumes significant time. Usage of DPI-C function gives important advantages in this aspect over use of static, pre-generated value tables. Any signal manipulation implemented in design should in theory align to its counterpart inside C model. When working with a C function integrated through DPI-C, a verification engineer can always access C source code. This opens the possibility of printing any values of interest from C code. In case C model and DUT are containing multiple subsequent operations done on input signal this can help the verification engineer to pinpoint erroneous behavior

in DUT much easier by printing multiple "mid results" in C code and comparing them to design values during debug.

D. Reconfiguration

Approach with DPI-C also allows reconfiguration during the simulation which can be viewed as an advantage over usage of static, pre-generated value tables. In case a module has any signals or registers which might influence the results of mathematical manipulation implemented in the DUT, they can be changed dynamically during the simulation by writing to C model and design in parallel. With static, pre-generated in/out value tables, this is not possible. In cases where verification process of the module needs to cover scenarios with reconfiguration on the fly, usage of C model through DPI-C integration gives easy solution to this challenge.

VI. CONCLUSION

As a result of a conducted case study and subsequent analysis of two described methods for C model utilization in functional verification process, it can be said that C model integration of DPI-C System Verilog function is a superior methodological solution in comparison to the use of static, pre-generated in/out C model value tables, but its application comes at a cost of simulation performance.

Usage of C code through DPI-C simply offers more tools to verification engineers. The ability to run reconfiguration during the simulation comes as a significant advantage. Also, the debug process is much more efficient with this approach and can save a lot of time and effort when applied correctly. This advantage becomes ever more significant with the increased complexity of the design when DUT can be divided into smaller units. Coverage collection also seems easier with solution using DPI-C. If simulation time is not a major concern and the number of modules which require C model use in verification process is limited, then integrating C model using DPI-C is probably a better solution.

Although it can be said that use of static, pre-generated in/out C model value tables is an inferior solution in this comparison, it can still be a more optimal choice for certain types of design. As comparative case study showed simulation time using this approach is significantly shorter. This means that if very long test cases are planned before verification is started and complexity of design under verification is high this solution is probably a better option where time saved in simulation surpasses any gains that integration of model through DPI-C offers in debug, coverage collection or ability to run reconfiguration scenarios.

Decision over which approach is more optimal to be applied will depend on specification of the design which is being verified but that doesn't mean that its optimal to switch between approaches over the course of a single project as in that case, the C model specialists need to support two different integration methodologies for the C model.

REFERENCES

[1] Universal Verification Methodology (UVM) 1.2 Class Reference, 2011 - 2014 Accellera Systems Initiative (Accellera)

[2] SystemVerilog 3.1a Language Reference Manual, 2004 by Accellera Organization, Inc.

[3] Parag Goel, Amiot Sharma, Hari Vinodh Belisetty, ""C" you on the faster side: Accelerating SV DPI based co-simulation", DVCon US, 2014

[4] MathWorks Verifiication and validation page - https://ch.mathworks.com/solutions/verification-validation.html

[5] Generator for CRC HDL code web page - https://bues.ch/cms/hacking/crcgen.html

[6] Xcelium XRUN User Guide - Product Version 19.09, September 2019

[7] Cadence Community Verfication web page: Best Practices to Achieve the Highest Performance Using Cadence Xcelium Logic Simulator – Part 1, 17 Jul 2023 - https://community.cadence.com/cadence_blogs_8/b/fv/posts/best-practices-to-achieve-the-highest-performance-using-cadence-xcelium-logic-simulator-part1

[8] [2] Chris Spear, Kevork Dikramanjian, Abhisek Verma, and Senay Haile, "C through UVM: Effectively using C based models with UVM based Verification IP", DVCon US, 2013