# Verifying Multiple DUV Representations with a Single UVM-e Testbench

Matt Graham
Applications Engineering
Cadence Design Systems
Ottawa, Canada
magraham@cadence.com

## I. BACKGROUND

As Systems on a Chip (SoCs) becoming increasingly larger, more complex, and further reaching in feature inclusion, designers and verification engineers continue to be challenged to develop and refine strategies to effectively develop and test their products. Terms like "reuse," "divide and conquer," and "advanced methodology" are regularly used, but their efficient application is not trivial.

Consider the case of a group of intellectual property (IP) developers creating some advanced processing IP within a large semi-conductor manufacturer. Groups such as this are commonplace, with the resultant IP processing everything from video and audio data (frames or samples) to digital network traffic (packets). Such groups are almost uniformly staffed with experienced design and verification engineers. Most employ advanced design and verification methodologies, and have an excellent track record of successful, first pass deliveries.

Despite all of this technology, experience and past success, IP development cycles are continually under pressure to improve. Increasingly complex IP must be delivered with equally high quality with decreasing resources, be they headcount, budget, time, or often, all three. Development teams are having to look to more creative and innovative solutions to solve their schedule and resource constraints while still producing high quality results.

## II. DEVELOPMENT FLOW

Recently, a group of video IP developers began exploring the option of automatically generating Verilog RTL based on previously developed C models of the processing algorithms. The development team was already producing C models to refine the various video processing algorithms to be included in the IP. The IP needed to be delivered approximately once every six months, with each delivery including new processing capabilities, increased speed and efficiency, and more data throughput than the previous generation. The team had been using the C algorithm models as part of the development process for some time, but up to this point had always coded the hardware representations of the algorithms by hand using Verilog. See Fig. 1.

Fig. 1. Traditional Development Flow

To be able to take advantage of the automated process for moving from C to Verilog, the C models had to be updated somewhat to include further implementation information. Thus the team had now created what they referred to as "Tool Ready C-models". See Fig. 2.

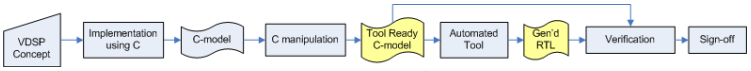

Fig. 2. Automated RTL generation development flow.

## III. PROBLEM STATEMENT

Despite being "Tool Ready" the C models were not implemented in SystemC. This was a result of a requirement of the C to RTL conversion tool rather than a specific design decision. Despite the automated nature of the conversion from C to RTL, delivery of usable RTL for verification lagged the delivery of the C models by approximately five weeks.

The verification team also determined that the entire final RTL sign off process must be accomplished using RTL simulations. While functional coverage can be collected from the verification environment for simulations using any representation of the device under verification (DUV), code or implementation coverage can only reliably be collected from implementation (synthesis, timing, etc, etc) flow. Finally, the team determined that there existed no acceptable flow for proving equivalence between the C models and the generated RTL. Existing C to RTL equivalence tools must "flatten" RTL, removing all synchronous logic (i.e. flip flops) in a logic tree. Equivalence is then only proven on the data transformation, rather than the synchronous circuit. It was determined that this was not sufficient for RTL sign off.

Based on the above requirements and determinations, the team decided that verification would need to be carried out on both

the C and RTL representations of the DUV. Verification of the C models gave the team as much as a five week head start on the verification effort before RTL was available. RTL verification was still required, as final sign off still needed to be completed on the RTL DUV.

## IV. GOALS

Having determined that both the C and RTL DUVs needed to be verified, the team set a number of goals for the effort. The first was to take advantage of the nature of the C models. Their earlier availability would help to get the verification effort started early, pipe cleaning the verification environment and further proving the validity of the algorithms. The higher abstraction level of the C models also could also help during early debug phase, making both simulation and debug more efficient.

To help realize the efficiencies stated above, the team also added a goal to utilize existing RTL verification IP as much as possible. This included everything from Universal Verification Components (UVCs), through functional coverage definitions, test writer interface and test flow.

It became clear at this point that the most effective path would be to architect and build a single Universal Verification Methodology (UVM-e in this case) environment, such that the same environment could be utilized to verify both the C and RTL representations of the DUV.

## V. VERIFICATION ENVIRONMENT – DUAL DUV

Development of the verification environment began by architecting a typical RTL verification environment, as had been used in the past. The DUV shown in figure 3 below includes a Verilog test harness, which replicated how the IP would be instantiated in the system. The test harness was also responsible for generating clocks, reset and providing all interfaces of the DUV to the verification environment.
The Config Structure generated, via constrained random stimulus generation, a valid configuration for the DUV, and created a sequence of commands for configuring the DUV (i.e. register reads/writes, processor commands, etc).
The Test Loop implemented a virtual sequence that defined the generic test flow. This flow is guided by the config structure, and also represented the framework within which the specific "test" is executed.

The Command Bus UVC was a standard UVM-e UVC for the protocol of the DUV command bus. In this particular case, this was a non-standard (internal) bus protocol, but generically could have been any protocol such as PCI, AMBA, etc. Other assorted UVCs were connected to all other interfaces of the DUV, to both stimulate and monitor those interfaces.

The monitor portions of the various UVCs captured data on various interfaces of the DUV and forwarded it to the scoreboard via analysis ports. The same data was also simultaneously passed to the predictor.

The predictor was a custom, cycle accurate model of the DUV. In many cases this predictor made use of the original C-model (not the "tool ready" C-model), but in other instances the predictor was hand coded by the verification engineers using the 'e' HVL. Output from the predictor was passed to the scoreboard via more analysis ports to enable end to end data checking.

A design goal for the verification environment was to maintain the same verification environment between the two representations of the DUV. As the environment was architected, it became clear to the team that having a single "DUV Select" control parameter for the verification environment, allowing easy switching between the C and RTL, would be ideal.

Abstracting the DUV representation from the verification environment via the test harness, as conceptually shown in figure 3, would also allow capturing waveform data at the output of the UVCs for simple debug. It would also allow for identical stimulus to be provided to the DUV by the verification environment regardless of which DUV implementation was used, for a given random seed.

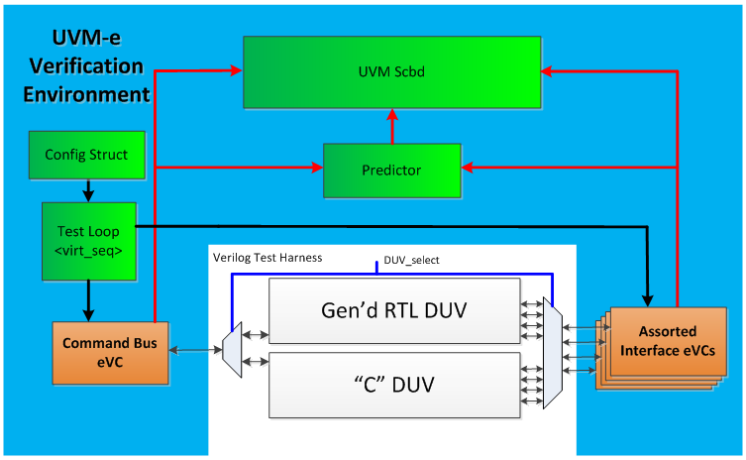

Fig. 3

## VI. C DUV Test Harness

To facilitate connection of the C DUV to the verification environment, the Verilog test harness had to be updated to include some sort of verilog to C converter. The C model, which had the notion of interfaces, did not include any timing information. The expanded test harness had to add timing information as well.

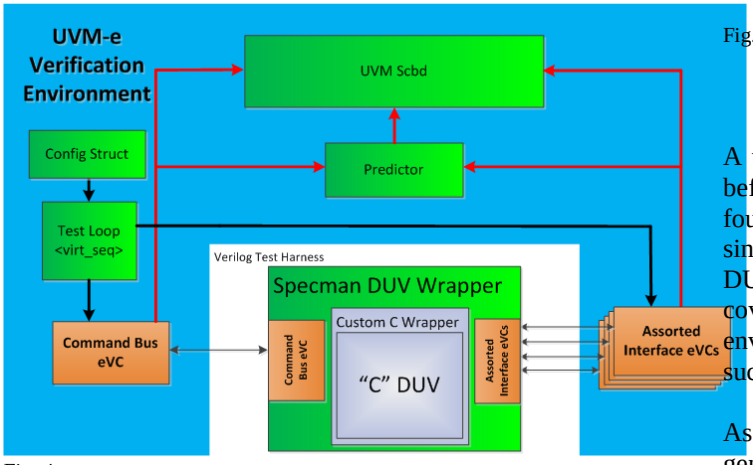


Fig. 4

The team utilized the C interface provided by 'e' to develop a wrapper for the C DUV. As the UVCs utilized in the verification environment already provided the client side of the various interfaces (as recommended per UVM), they were easily used to abstract the stimulus data up to the level required for the C DUV. See Fig 4.

A single C model thread executed from the beginning of simulation time. The ports of the DUV acted as FIFOs, defined in varying lengths depending on the specification of each interface. Each occurrence of a put or get had the C model thread execute a call back to the e DUV wrapper. The wrapper would then stall until the data was received or sent.

The alternative approach, to bypass the signal level interface of a UVC and connect directly to C code, was considered. The team felt this approach was superior in cases where UVCs are new development. In this case, however, the existing UVCs required little or no rework from a functional standpoint. In addition, the client side portion of the UVCs were also already developed and proven. Thus the decision was taken to instantiate the client side UVC agents as a portion of the "Specman DUV Wrapper." The only new development was the interface from the client side UVCs to the C DUV (See Fig 5).

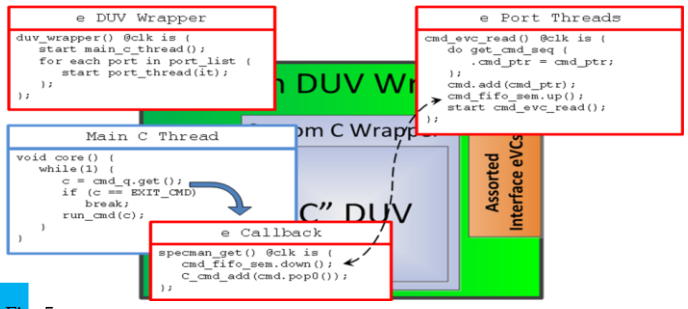


Fig. 5

## VII. Results

A total of 10% of the overall feature coverage was achieved before any RTL was available. 25% of all design bugs were found and fixed during the C model verification phase. The single environment was successfully used for all C and RTL DUV verification, including all stimulus generation and coverage capture. Once up and running, the verification environment needed no further changes when RTL was successfully generated.

As this was the first pass through the automated RTL generation flow for this team, a number of challenges were encountered, and lessons taken away. The team found that live debugging of the C model was troublesome. The team worked around these difficulties by capturing stimulus from the UVM-e verification environment and simply playing it back through the C model while debugging with GDB. The team felt that some time spent with simulation tool support would alleviate this problem in the future.

The e-to-C DUV wrapper required a significant initial investment, nearing 4 person weeks. It was expected that future utilizations of such an environment would be much lower cost. The aspect oriented nature of e should enable rapid reuse of the wrapper on future projects.

The team also felt that further investigation into the usage of SystemC for the C models would be warranted in future projects. SystemC would enable timing information to be embedded in the C DUV, thus simplifying the wrapper. Simulation tool support for SystemC might yield better native debug support as well. Utilization of standard SystemC TLM interfaces might further simplify the C DUV wrapper as well. The team found that the ability to dynamically select between DUV representations was a major benefit to the architecture and were keen to maintain this feature.

Future passes through this development path would likely result in greater percentages of functional coverage achieved. Shorter development cycle of the e-to-C wrapper could result in more time for simulations, and greater collection of functional coverage. The main factor limiting functional coverage collection during the C DUV phase of this verification effort was time.

## VIII. SUMMARY

The verification environment architected to allow verification of both C and RTL representations of the DUV helped the verification team get access to the DUV a full five weeks before RTL was available. Despite the need for complete verification sign off to be accomplished with the RTL DUV, early access to the C DUV enabled the verification team to achieve 10% functional coverage and find 25% of the design bugs before RTL availability.

The e-to-C DUV wrapper developed enabled the team to utilize a single environment for both DUV representations, and utilized the existing library of Verification IP available to the team.

Despite some debug tool issues, the entire development team viewed the project as successful.