Bridging the gap between TLM-2.0 AT models and RTL – Experiments and Opportunities

Zhu Zhou, Atul Kwatra, Rajesh Gadiyar, Paul Heraty
Intel Corporation
5000 Chandler Blvd., Chandler, AZ U.S.A
zhu.zhou@intel.com
atul.kwatra@intel.com
rajesh.gadiyar@intel.com
paul.heraty@intel.com

ABSTRACT
As the transaction level modeling methodology gets widely adopted, the fidelity of TLM accuracy becomes the key in the performance assurance process. This paper details one approach that reuses the TLM test environment to exercise both TLM and RTL DUT to ensure their consistency. Some additional opportunities of filling the gap between TLM and RTL models are suggested.

Categories and Subject Descriptors
D.3.3 **[Programming Languages]:** Language Contructs and Features – *abstract data types, polymorphism, control structures.*

General Terms
Performance, Design, Standardization, Verification.

Keywords
Performance Modeling, TLM-2.0, Approximately-Timed

1. INTRODUCTION
As design complexity keeps increasing, improving system performance within a certain power envelope mandates platform architecture exploration at a higher abstraction level in the early design stages. Performance modeling has been successfully used for early architectural studies. The OSCI TLM-2.0 standard further accelerated the adoption of SystemC based transaction level modeling methodology by enabling interoperability and reusability. However, assuring performance projections (throughput and latency) through the process of RTL implementation remains a big challenge.

In this paper, a mixed TLM-2.0 AT style model and RTL simulation infrastructure is introduced with the goal to validate performance aspects and to identify RTL performance bugs earlier. Most commercial simulators support mixed SystemC and HDL simulation thus enabling reuse of the SystemC test bench developed during the performance modeling phase to verify RTL implementation within the platform context. The simulated throughput and/or latency discrepancy between SystemC and RTL can result in either earlier RTL bug fixes or micro-architecture modifications (along with corresponding modifications to the TLM model). Performance analysis can be done at the system level to justify the design impact.

It is worth noting that we are referring to consistency from a performance perspective here; throughput and latency. Functional equivalence checking between SystemC and RTL is a separate issue that is not addressed in this paper.

We also further discuss other opportunities to bridge the gap between TLM models and RTL implementations.

2. PERFORMANCE MODELING STAGES
Performance modeling plays different roles during the whole design process (Figure 1). At the early architecture definition stage, performance modeling serves as a sand box for architects to weigh different options such as software/hardware partition, interconnect protocol, IP choices, etc. Once the overall platform architecture is defined, modeling the detailed micro-architecture of each block can often refine/optimize the system performance. The Approximated-timed coding style defined in TLM-2.0 standard suits very well for these two tasks given its timing granularity, flexibility and simulation performance. Processes in the AT coding style run in lock-step with simulation time in order to provide the accuracy needed for throughput/latency projections.

Today, most of the implementation starts with hand written RTL (e.g. SystemVerilog). The purpose of pre-silicon RTL performance validation is to ensure that no bugs are introduced to negatively impact the projected throughput/latency. Note that the refinement to the models continues as the design proceeds to the next level. The detailed information goes back to validate early assumptions and change the performance projections if needed. This includes correlating post-silicon performance measurement with high level performance models that is not shown in the diagram.

Correlating TLM-2.0 AT models and RTL implementations is difficult and time-consuming because they are created independently by different teams. On the other hand, it is crucial to fill the gap and ensure two set of modules are consistent.

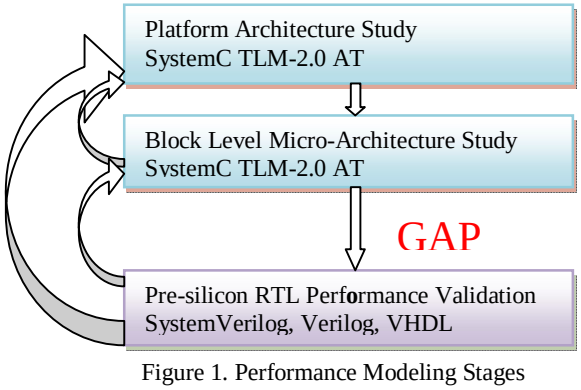

Figure 1. Performance Modeling Stages

3. PERFORMANCE CORRELATION BETWEEN TLM-2.0 AND RTL

A real project is used to prove the reusability of the whole SystemC test environment for newly developed IP and correlate the TLM-2.0 AT model with the RTL implementation. There are certain challenges with integrating TLM-2.0 models with RTL. The chosen approach is described and the resulting impact on simulation speed is summarized.

3.1 TLM-2.0 AT Style Performance Model

Figure 2 (except the light yellow area) shows a performance test platform to ensure that newly developed IP can support required throughput and latency under the system context. All modules were developed using the TLM-2.0 AT style. The detailed pipeline, buffer size and arbitration scheme are modeled to predict actual performance under various configurations. Traffic generators are configured to issue memory accesses representing certain real usage scenarios. The platform has been simulated extensively to identify bottlenecks and optimize the platform level configuration.

The high level performance study is conducted before RTL coding gets started and the models/configurations continue to be refined during the RTL development.

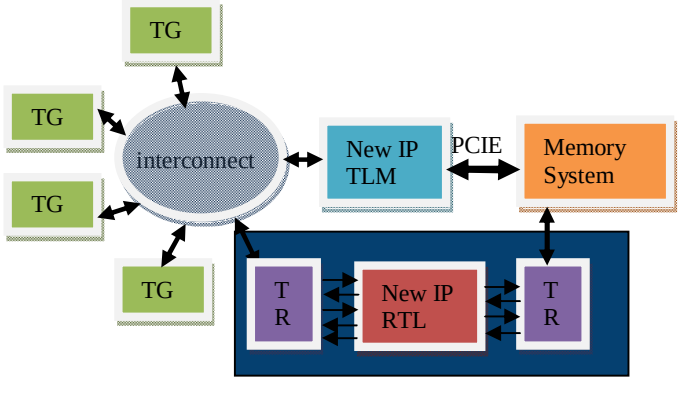

Figure 2. Mixed TLM-2.0 and RTL test platform

3.2 RTL Performance Assurance

Once the RTL code reaches a certain level of stability, it is important to validate the predicted throughput/latency by applying the same test scenarios from TLM-2.0 test platform. As shown in figure 2, the transaction flows through the lower path to exercise RTL implementation. In order to interface TLM models with RTL, transactors (shown as TR in the figure) are needed to translate the high level function calls into RTL level signals and vice versa. Transactors can be coded in either SystemC or RTL (Verilog, VHDL, and SystemVerilog). In our case, SystemC is used and a commercial RTL simulator provides a SystemC wrapper for the RTL DUT so that it can be connected with the transactor in the top-level netlist.

TLM-2.0 transaction objects are created in the traffic generator and passed on to other modules along the memory path using reference pointers. One issue with this is that when TLM-2.0 transaction crosses the transactor into RTL, the pointer to the TLM-2.0 transaction object is lost. This means that a new TLM-2.0 transaction needs to be created in the transactor that connects to the memory system using the information passed out from the RTL. Also for memory reads, the transaction pointer needs to be stored within the transactor. When the associated completion comes back, it can be retrieved to model the path that goes back to the original initiator. Memory management of TLM-2.0 transaction objects needs to be carefully designed to handle such situations.

The TLM-2.0 based modeling infrastructure is not intended to be used as the comprehensive functional test environment. However, the RTL coders did find this platform based environment useful for running quick unit level tests before handing off code to the pre-silicon validation team, particularly given the fact that it uses the same RTL simulator and debug tool suite.

3.3 Debug RTL and TLM in Parallel

It is common to find performance discrepancies between pure TLM platform and mixed TLM/RTL simulation. The causes can be bugs in either the RTL or TLM models. To root cause the problem, debugging TLM and RTL models simultaneously with same injected traffic is desirable. Activating both paths in Figure 2 poses some challenges in terms of handling TLM-2.0 transactions. For example, the traffic generator (TG) creates a new TLM-2.0 read transaction. It goes through the interconnect and splits into both the TLM and RTL IP models. Likely the latency differs, so they arrive at the memory system module at different times. The memory System module needs to be modified to handle this situation. In addition, two data completion paths have the same issue when they converge at the interconnect. Even more challenging, the single extended TLM-2.0 payload object will likely be modified by two parallel paths at different time.

To make things easier, the same set of test bench components is instantiated twice to separately drive the TLM model and corresponding RTL as shown in Figure 3. The traffic generators are configured the same to ensure the same traffic gets injected into both paths. The TLM-2.0 analysis ports are used to export information at various points to a performance correlation module where various checkers/monitors can be instantiated to log/report any performance differences between the TLM and RTL paths.

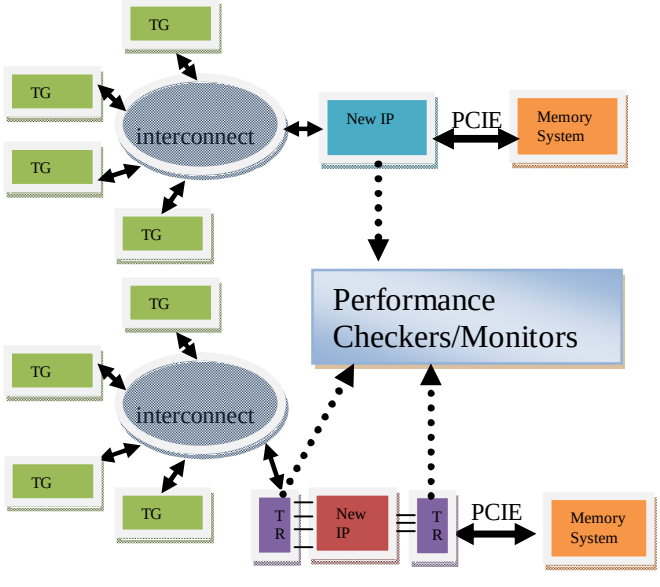

Figure 3. Parallel TLM-2.0 and RTL test platforms

3.4 Simulation Speed Comparison

One main reason of moving to higher abstraction levels is to achieve several magnitudes higher simulation speed. With that, extensive architectural exploration can be conducted within a reasonable time

frame. Table 1 shows the speed difference between pure SystemC simulation and SystemC/RTL mixed simulation.

Table 1. Simulation Speed Comparison

Platform	Simulator	Speed (transactions/second)
Pure TLM-2.0	OSCI simulator	50, 000
Mixed TLM-2.0/RTL	Commercial RTL simulator	64

4. OTHER OPPORTUNITES

Similar to RTL design, validating TLM models is much more time consuming than writing the model. In order for TLM modeling to become part of the mainstream design flow, the gap between a TLM model and its RTL implementation needs to be filled. Some other potential approaches are discussed below.

4.1 High Level Synthesis (HLS)

High level synthesis tools have gained momentum in recent years. The focus has been the quality of generated RTL code. There is a potential opportunity in having the HLS tool generate an equivalent TLM 2.0 AT model in addition to synthesizable RTL from the high-level design description (Figure 4). This TLM model can then be plugged into the platform level simulation environment to quickly validate the performance of the implementation choice in the full system context. The key technical challenge in generating the TLM 2.0 AT model would be to abstract away appropriate RTL functionality and timing details to achieve faster simulation speeds. Today most synthesis tools can generate signal level cycle accurate SystemC representations. In order to plug this model back into the TLM-2.0 platform, transactors need to be built and simulation performance may be a concern given the extra detail included in the cycle accurate model.

Additionally, the generated RTL can be synthesized for power estimation. The power information can be fed back to the TLM platform to do performance/power trade off analysis.

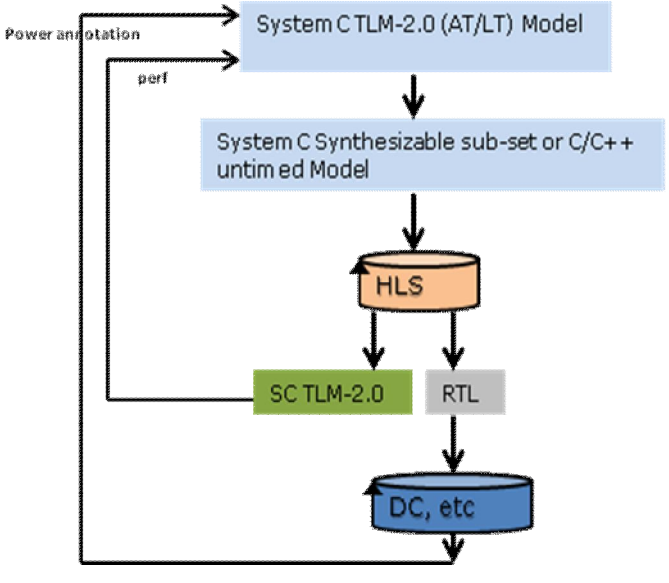

Figure 4. Ideal High Level Synthesis design flow

4.2 Handling Legacy RTL Modules

IP reuse has become a popular practice to improve time to market. One critical task of SoC design is to properly assemble/configure available IPs (include interconnects). Transaction level modeling has been successfully applied for performance, power and area trade off analysis. For existing RTL modules, it is time-consuming to hand write and validate a corresponding TLM model.
Commercial solutions are available to plug existing RTL models into a virtual platform by converting RTL into C/C++ code or by abstracting the timing information required by the high level platform. The challenge is to balance accuracy, visibility, and simulation speed. Also additional effort is needed to handle the interface through transactors.

4.3 Reuse of Transaction Level Checker and Monitor

For a design flow that starts with TLM performance models, transaction level protocol checking mechanisms have been used on both interconnects and internal pipeline logic. It is desirable to standardize/reuse/translate such high level checkers into RTL assertions that can automate the correlation process and dynamically check the correctness of certain protocols. Transaction level visualization is another important tool to debug the performance aspect of the platform behavior. Some form of correlation between transaction view and RTL signals will certainly help quickly identify discrepancies between TLM and RTL.

5. CONCLUSION

It becomes impossible to design a competitive chip architecture using traditional spreadsheet based performance analysis. TLM based performance modeling is a necessary tool for architects to explore different design options at a very early design stage. Because critical decisions are made based on the simulated platform, the TLM models have to be validated against the specification and later serve as a golden model for the RTL implementation. Correlating RTL with TLM model is usually a huge effort. Since the TLM-2.0 standard is relatively new, there are almost no automation tools to cross-check a TLM-2.0 model and its corresponding RTL implementation from the performance perspective.

This paper has epitomized how we can implement both the TLM and the RTL DUT within the system context to identify any discrepancies. This helps to flush out any RTL bugs that impact the system performance. In addition, several other techniques that have potential to fill the gap between TLM-2.0 model and RTL are discussed. Hopefully, some commercial tools will start to mature in the near future to make the top down design flow smoother.

6. ACKNOWLEDGMENTS

We would like to thank the following co-workers at Intel who have supported the related projects and provided valuable inputs in defining the methodologies.
Qi Zhu, Trevor Wieman, Brian Mears, Asad Khan, Kaushik Bhatt, Michael Kishinevsky

7. REFERENCES

[1] OSCI TLM-2.0 User Manual, Open SystemC Initiative (OSCI), Version JA22, June 2008.
[2] IEEE Std 1666-2005, IEEE Standard SystemC Language Reference Manual. March 31, 2006
[3] OVM SystemVerilog User Guide, Version 2.0.2, June 2009.

[4] Frank Ghenassia. Transaction-Level Modeling with SystemC : TLM Concepts and Applications for Embedded Systems. Springer, 2005.

[5] Thorsten Grotker, Grant Martin, Stan Liao, Stuart Swan. System Design with SystemC, Kluwer Academic Publishers, 2003.

[6] SystemC Synthesizable Subset 1.3 draft. June 30, 2009 OSCI