2016
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
EUROPE

TLM Beyond Memory Mapped Busses

Bart Vanthournout, Synopsys

Mark Burton, GreenSocs

The TLM-2.0 standard has proven highly valuable to the SystemC ecosystem but optimally supports only memory mapped bus communications. Using TLM-2.0 for other model-to-model interfaces such as interrupt signals, serial protocols, and Ethernet-like protocols, was never intended and doesn't work very well. The alternatives are to create custom interfaces, complicating interoperability, or to use SystemC core interfaces like sc_signal that have semantics largely incompatible with TLM. The TLM WG has set out to extend TLM-2.0 to more comprehensively address model-to-model communications.

Keywords—TLM; SystemC;

I. INTRODUCTION

Since the publication of the original TLM-2.0 standard in 2008 the use of Virtual prototyping has increased significantly. Over the years many design teams have adopted the TLM modeling technology and standard for the creation of Virtual Prototypes used for software development and optimization, architecture exploration and verification use cases, and all this for an increasing number of application domains. This is leading to a number of new challenges: as the complexity and range of designs is increasing, so is the number of interfaces that need to be modeled. While TLM-2.0 has greatly assisted in bringing a standard approach to the memory mapped bus interfaces within a system model, the same cannot be said for non-memory mapped bus interfaces. Different organizations have solved this issue in different ways, and this presents an interoperability issue: different models use different approaches and different tools support different approaches. In general, unfortunately, the result is poor support for serial interfaces, and confusion within the SystemC community. In this paper the challenges that are being raised with regards to the TLM-2.0 interfaces are discussed. These are then put in perspective with the original goals and implementation of the standard, and a few proposals to resolve these issues are described.

II. REQUIREMENTS

Several documents have been written about requirements, (and submitted to the TLM Working Group). They cover, in detail, the different aspects of different interconnects, the different modes those interconnects can work in, and therefore the requirements on what any 'TLM' standard should support. What becomes apparent is the vast diversity of interface mechanisms. From a standardization point of view there are a number of interesting interfaces to look at that highlight the challenges for the TLM-2.0 standard

A. Support for interrupts

In a prototype where TLM-2.0 is used for all memory-mapped interfaces the next candidate to look at are interrupts since they closely relate to the SW execution. But in general we can consider all simple wires interfaces between components. The SystemC standard provides simple sc_in and sc_out classes to model wires but, even though they are used in many TLM systems, these are not compatible with the TLM-style for modeling. In and out ports cannot be connected with each other, it is required to insert an sc_signal channel. More importantly, the semantics for an sc_signal demand the use of delta-cycles and signal-resolution to propagate data between the two ports. This can cause synchronization issues in a TLM-2.0 based prototype using quantums.

B. Support for GPIO

For General Purpose IO device (GPIO) interfaces it is often the case that the actual protocol is programmed through software, and could be altered. It is therefore important that the impact of the software is modeled in the TLM prototype, including the ability to alter GPIO interfaces to operate different protocols. However, this could mean that the ‘transactions’ present in a protocol are lost. Not only is this potentially a performance issue, it also increases the complexity of a model, and the understanding of how models should be written.

C. Support for SPI

SPI is just one example of an interface which is commonly used to transfer high volumes of data, or complex frame structures over a serial interface. In many cases this data appears in the memory map of a processor, or it is used to access ‘addressed’ registers in the targeted device. In this case the serial interface looks similar to a memory-mapped bus but with different transactions, command structure, timing etc.

III. CHALLENGES WITH THE TLM-2.0 STANDARD

The requirements expressed above expose a set of issues with the TLM-2.0 standard. These and related problems with the standard have been explicitly raised and documented in reports to the TLM-WG. The issues include, but are not limited to:

- The generic payload and its extension mechanism are not sufficiently adaptable for many interfaces. The key is not with the extensibility but with the standard attributes of the payload. The standard does not provide any mechanism to exclude payload attributes, nor does it provide any guidelines on how to construct new payloads.
- The standard does not contain sockets that support duplex communication as is so often required by serial protocols.
- The focus on using blocking interfaces for models supporting the SW development use cases is not always ideal, there are cases where non-blocking semantics could be much more convenient. However, the non-blocking interfaces mandate the use of a phase argument, which has no relevance for some use cases.
- Sometimes there is no need for a backward path for the non-blocking interfaces
- Provisions for Direct Memory Interface are not applicable for some protocols
- There are issues with hierarchy support for the sockets.
- It is not obvious how to deal with reset in a TLM system.
- TLM-2.0 comes with a large and complex set of rules, ideally these could be relaxed for simple interfaces.
- Certain modeling styles need to know the binding path between 2 sockets post elaboration
- The TLM standard did not discuss or address support for HDL co-simulation.
- The standard forces strict interoperability while in many cases models of different coding styles or abstraction levels must live together in a design
- The quantum-based synchronization mechanism that is introduced in the standard does not allow targets to influence the synchronization rate other than through the SystemC kernel.

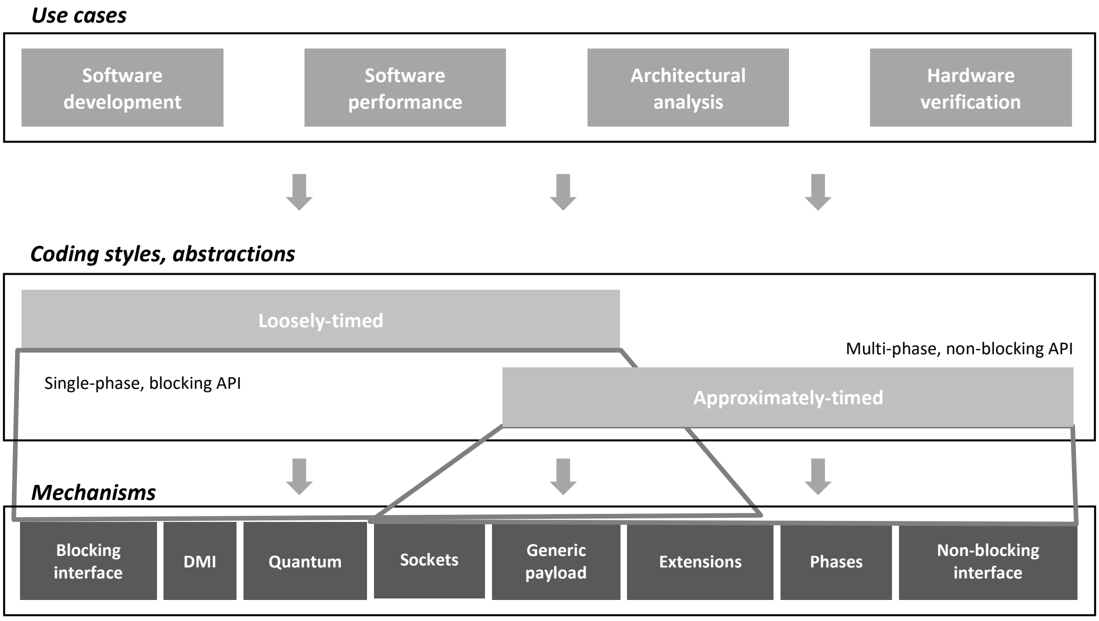
For all of the above issues the standard does not provide with solutions or a guideline to address the issue, as a consequence multiple approaches to address these issues are in use, degrading potential interoperability.

IV. THE TLM-2.0 STANDARD REVISITED

It is clear that the list above poses major challenges to the TLM-2.0 standard. This is a consequence of the fact that TLM-2.0 is created with memory-mapped busses in mind. The TLM-WG chose this focus: memory mapped buses are the key to simulation performance, which was the most important challenge to address. As a result, the standard comes with a generic payload definition that is fully focused on memory mapped interfaces. There are two interface styles: one optimized for maximum performance, another for use cases that require extra timing detail.

The direct memory interface could be considered as a third API for maximum performance, also designed with memory mapped busses in mind.

Because TLM-2.0 only defines a payload for memory mapped buses the model interoperability is limited to those types of interfaces. When the standard was released there was an expectation with the TLM-WG that protocol-owners would use TLM-2.0 to define protocol specific customizations, however that never materialized. As a result, there is a perception that the TLM-2.0 standard only supports memory mapped busses. However, that perception is partially unfounded. There is an aspect of TLM-2.0 that is designed as a set of building blocks which can be used to construct different protocols for multiple coding styles and abstractions. This is best exemplified through the picture below which was used when introducing TLM-2.0. It shows the layered approach of use cases, coding styles and 'mechanisms'.



The focus of the TLM-2.0 standard has been on the coding styles and abstractions applied to a specific protocol (the Base Protocol), little guidance is provided for modeling protocols that require different coding styles or abstractions. However, if we take the individual elements, building blocks or mechanisms as they are called in the picture above there are sufficient features that are well defined and well known throughout the community to serve as a starting point for extending SystemC-TLM modeling beyond its current scope.

V. MOVING FORWARD

Given the range and multitude of issues listed in Section III it will not be sufficient to simply highlight the existing underlying API's in the TLM-2.0 standard and leave it to the users to figure out how to address the requirements listed earlier in this paper. Much more will be needed. As [1] has suggested, a proposal to structure the workflow and address the issues could be as follows

A. Improve the 'Mechanisms'

Part of what needs to be done is to revisit the different mechanisms and upgrade them based on the issues listed above, some of this is pure 'bug-fixing' or enhancements of the existing implementation and some requires rework the the mechanisms. On top of that a discussion is needed as to whether additional mechanisms are required. Example topics are:

- Payload refactoring: the current TLM-2.0 standard does not provide a truly 'generic' payload, as mentioned before it is very much memory-map centric. So there is a need to guide users to construct their own payload in such a way that conversion between protocols or interfaces is easy (e.g. from GPIO to SPI). There is also a need to make it easy to create 'frame'-based payloads.

- Rework the interfaces: there may be a need for more flexible core interfaces but more importantly there is a need for additional interface groups to address the need for non-blocking only interfaces
- Additional sockets: so that duplex and other interfaces can be modeled
- Introduction of annotated time scheduling: create a closer interaction between the timing annotation in the interfaces and the quantum that is driving initiator scheduling.

The goal for these mechanisms is to provide a common vocabulary for creating TLM interfaces. When all TLM interfaces are built on the same semantic constructs it will be much easier to train modelers on their use and it will make it easier to create and define new TLM interfaces.

B. Introduce new coding styles and protocols

When extending the SystemC TLM standard it is important not to get lost in the details of the mechanisms. The enhancements and extensions to the mechanisms need to be driven from a definition of additional coding styles and protocols. New coding styles will need to be introduced for non-memory mapped interfaces. A common ground will need to be found for groups of interfaces in order to avoid that each new interface will come with its own unique combination of TLM mechanisms. A key driver for this activity is to ensure ease of use and to reduce complexity. These additional coding styles can then be used as guidelines for the definition of specific protocols. Moreover, it might be required that the TLM-WG changes its stance towards specific interface definitions. So far the group did not wish to standardize such interfaces, and that should probably largely remain as is, but for certain protocols the group might be in a position to be able to standardize a specific TLM interface. This will be important in order to validate whether the improvements to the standard works well for real protocols, and also as a jump start to invite others to do the same.

C. Define guidelines and 'blue-print'

Finally, there is a need for a set of guidelines for the creation of custom interfaces and specific protocols using SystemC-TLM. There have been suggestions and proposals for this in the TLM-WG in the past. The 'blue-print' should guide the definition of an interface in a way so that it will be consistent. Indicating how to identify 'transactions', how to encode the features of an interface, how to document an interface and what other material to provide with an interface kit.

VI. CONCLUSION

The memory map centric definition of TLM-2.0 is hampering the effectiveness and interoperability of SystemC model development for state-of-the-art systems. This paper presented an overview of the issues that have been encountered when applying TLM-2.0 to non-memory mapped bus interfaces. As a result, the TLM-WG has captured requirements, explored solutions, and proposes to tackle these challenges through a three-pronged approach: going back to the basics of TLM modeling and improve, enhance and extend the mechanisms defined in TLM-2.0; introducing additional coding styles and protocols that meet the new requirements; and providing a guideline, or blue-print, that can be used to create TLM customizations that are consistent with the standard.

REFERENCES

[1] G. Delbergue, M. Burton, B. Le Gal, and C. Jego, "Analysis of TLM-2.0 and it's Applicability to Non Memory Mapped Interfaces," in Design & Verification Conference and Exhibition 2016 (DVCon16), San Jose, USA