Fast and Accurate Protocol Specific Bus Modeling using TLM 2.0

H.W.M. van Moll, H. Corporaal
Technical University Eindhoven
Eindhoven, The Netherlands

V. Reyes, M. Boonen
NXP Semiconductors
Eindhoven, The Netherlands

Abstract—The need to have Transaction Level models early in the design cycle is becoming more and more important to shorten the development times of complex Systems-on-Chip (SoC). These models need to be functional and timing accurate in order to address different design use-cases during the SoC development. However the typical issue with Transaction Level Modeling (TLM) techniques is the accuracy vs. simulation speed trade-off. Models that can run at high simulation speeds are often modeled at abstraction levels that make them unsuitable for use-cases where timing accuracy is required. Similarly, most models that are cycle accurate are inherently too slow (due to clock sensitive processes) to be used in use-cases where high simulation speed is key. This paper introduces a new methodology that enables the creation of fast and cycle accurate protocol specific bus-based communication models, based on the new TLM 2.0 standard from the Open SystemC Initiative (OSCI).

III. METHODOLOGY

Virtual prototyping (VP) technology based on SystemC TLM models are rapidly settling down in the semiconductor and EDA industries as a solution to cope with the increasing design complexity and shorter time-to-market of today’s SoC. VP can be applied successfully on different design use-cases along the SoC development cycle. Some VP use-cases, such as SW development or functional verification, have simulation speed as their main requirement, since millions of instructions have to be executed during simulations. Other use-cases, such as performance verification, have cycle-accuracy as the main requirement, since the obtained results are used to sign-off the HW platform. However, as of today, TLM techniques cannot create cycle-accurate models that are fast enough to fulfill the speed requirements of all VP use-cases.

Another concern regarding TLM is the lack of a standard for creating the models. This has led to many different implementations that are generally not compatible with each other [2][3]. Recently, OSCI has tackled this problem by proposing TLM 2.0 as the interoperability standard [1]. The OSCI TLM 2.0 standard addresses several of the shortcomings of the TLM 1.0 standard with respect to model interoperability and simulation speed. The standard focuses on SoCs that are based on memory-mapped buses and defines two coding styles: loosely-timed (LT) and approximate-timed (AT). However, no coding-style for cycle-accurate (CA) modeling is defined in the TLM 2.0 standard.

The ideal goal is to combine the accuracy of RTL models with the speed-up that TLM models provide using the TLM 2.0 interoperable standard. The methodology presented in this paper contributes to that goal by:

• Demonstrating that the TLM 2.0 standard provides the necessary mechanisms to create protocol-specific Bus Cycle Accurate (BCA) models.

• Providing a structured way to create protocol-specific BCA TLM 2.0 interfaces and transactors.

II. RELATED WORK

TLM has been successfully used in design tasks ranging from embedded SW development to functional verification. The requirements of the TLM model depend on the use-case for which it is needed. Cai and Gajski [2] present an overview of six models that vary both in the abstraction level of communication and computation from completely un-timed to fully cycle-accurate. The models that are created with our methodology have a close resemblance to the bus-functional model of [2] but using standard SystemC as the modeling language.

For the exploration of on-chip communication performance, models which have a high degree of timing accuracy are needed. In [3] Pasricha et al. present a transaction-based abstraction level called Cycle Count Accurate at Transaction Boundaries that is shown to have a simulation speed-up of 55% over traditional cycle accurate models, although lack of support for pipelining and out-of-order transactions in these models limits the accuracy for real protocols such as AXI or OCP.

Some IP providers have developed their own cycle accurate TLM APIs. Examples are IBM’s CoreConnect TLM models [4], the OCP-IP SystemC channels [5] and the ARM RealView APIs [6]. The problem with these implementations is that they do not follow a single standard and are therefore not interoperable with each other. The work shown in this paper contributes to extend the scope of the TLM 2.0 standard by using its extension capabilities to create bus-cycle-accurate protocol-specific models.

III. METHODOLOGY

The focus of this methodology [12] is to create a custom interface that enables BCA communication for a given protocol using the mechanisms provided by TLM 2.0. An important part of this process is to identify and create extensions to the generic payload data structure and phases defined in the standard [1]. Furthermore, this methodology proposes a structured way of creating the transactors needed to connect this BCA interface to a generic high-level interface that is used by the IP models. This generic interface allows separating the IP behavior from the interface details in the models, and hence enables reusing the same behavioral model for different protocols and/or modeling styles. In this work we have chosen the generic high-level communication interfaces provided by the SystemC Modeling Library (SCML) from CoWare [8].
The proposed methodology is composed of three distinct steps: Protocol Analysis, TLM 2.0 Mapping and Transactor Creation. Each step is split further into one or more tasks, as shown in Figure 1.

The first step in the methodology is the analysis of the protocol that is to be modeled. There are three relevant aspects of the protocol that need to be obtained: protocol attributes, timing points and the state transitions. Protocol attributes (signals) that are related, typically, are ordered in groups. Timing points indicate where certain signals become valid and are used as synchronization points between the master and slave. Finally, state transitions capture the behavior of the protocol. The second step of the methodology is the mapping of the protocol attributes, timing points and state transitions to TLM 2.0 structures. Attributes should either be mapped directly to generic payload (GP) attributes (preferred) or to custom payload extensions based on the extension mechanism provided by the standard. Timing points are translated into extensions of the generic payload phases. In this paper we define a channel as a set of attributes related to a timing point. Protocols can have one or more channels. The output of this task is a pair of custom initiator and target TLM 2.0 sockets that form the initiator and target bus interfaces of the transactors. Another action carried out in this second step is the mapping of the protocol channels to the state transitions defined in the first step. This task results in a set of finite state machines (FSM) for the master and slave transactors. The final step is the creation of the transactors that enable the connection of the TLM 2.0 BCA protocol interface to IP models with the generic high-level interface. When implementing the transactors, the output of the previous step is used to complete a transactor template both for the master and slave transactor. SCML and other proprietary libraries are used to create the generic interfaces and to model the internal of the transactors.

These steps are meant as general guidelines that apply to most memory mapped bus protocols. Examples of protocols that are in the scope of this work are, for instance, AMBA AHB and AXI from ARM [9], OCP from OCP-IP [5] and the DTL and MTL protocol from NXP[11].

The following sections will discuss each step of the methodology in more detail. The steps are further explained with the help of a protocol example, specifically the DTL protocol. The DTL (Device Transaction Level) protocol is a point-to-point data communication protocol that can be used for either direct IP to IP communication or IP to bus connections. It is comparable in functionality and complexity to AXI and OCP protocols [5][9].

IV. PROTOCOL ANALYSIS

A. Identify Protocol Attributes

The goal of this task is to examine the protocol signals and group related signals together. These signal groups will form the basis for the TLM 2.0 generic payload attributes and extensions in the mapping step. The output of this task is a set of protocol attribute groups.

For the DTL protocol the identification of attribute groups is simple since the protocol specification already defines a set of six signal groups. A total of 4 attribute groups are finally implemented in the TLM 2.0 DTL interface. The clock and reset are global system signals and are not part of the TLM interface (clock sensitive process are not used in the implementation) and therefore the “system group” is not considered. Signals from the “error/abort group” can be easily merged with the other groups and hence no separate group is needed for those signals either.

B. Identify Timing Points

Timing points are defined as the moment where a signal or a group of signals becomes valid and can be sampled by the receiver. In principle, a timing point could be identified for each signal transition of the protocol. However, only a few are actually required to model the protocol with the desired degree of accuracy. Only those signal transitions that actually mark a point in time where information is being transferred or where the state of the protocol is changed are of value as timing points. Identifying any other points would only increase the complexity of the model, while not adding more accuracy.

Identifying the timing points for the DTL protocol is very straightforward since the protocol uses handshake signals to indicate when signal groups become valid and when the receiver has sampled the signals. This means that the timing points for the DTL protocol can be aligned to the rising edges of the handshake signals. Figure 2 shows how the timing points for the command and read groups are identified. For the write and buffer management groups, similar timing points are identified. A total of 8 timing points (2 per signal group) are defined for the DTL protocol.

Figure 2: Timing points for the DTL Command and Read groups.
C. Identify State Transitions

Creating a FSM that captures the behavior of the communication protocol can be quite difficult, especially for complex protocols with many timing points and signal groups. It is therefore wise to limit the number of timing points and signal groups that were identified by the previous tasks to an absolute minimum, yet following the protocol specifications. Doing so simplifies the design of the protocol FSM and the other steps of this methodology, without necessarily hampering the timing accuracy.

The bases for designing the FSM are the timing points of the protocol and how they relate to each other. Identifying the possible flows of timing points that are supported by the protocol helps creating the FSM. At this point the state machine can be described in a relatively abstract way. Further details are added later, in the TLM 2.0 mapping step.

V. TLM 2.0 Mapping

The first task of this step is to map attribute groups and timing points together and create the required TLM 2.0 extensions. These custom data structures (channels) together are used to create the protocol specific TLM 2.0 interface sockets. The second task is the creation of the master and slave state machines.

A. Protocol Specific BCA Sockets

The TLM 2.0 non-blocking transport interface implements the \textit{nb\_transport\_fw/bw} method [7] that takes three arguments: a transaction object, a phase object and a time object. The transaction and phase objects can be customized to allow protocol specific modeling. The default generic payload transaction object will be extended with payload extensions to enable the transfer of protocol specific attributes and the phase object will be replaced with a custom enumerator that captures the timing points (phases) of the specific protocol.

If the timing points of the protocol were identified correctly there will be a direct one-to-one mapping of the timing points to transaction phases. Each timing point corresponds to one transaction phase. The phase of a transaction indicates which attributes of the generic payload and/or payload extensions are considered valid. A custom enumeration object must be defined that captures all of the transaction phases of the protocol. When mapping the protocol attributes to payload extensions it is often a good idea to map each signal group to its own separate extension. This supports protocols with pipelining capabilities, because each extension can be processed and routed separately from the other extensions. The custom payload extensions and custom phase object are used to implement a protocol specific pair of initiator and target TLM 2.0 sockets.

There were 8 timing points identified for the DTL protocol in the previous step. These can be directly mapped to 8 transaction phases. Figure 3 shows the phases that have been defined.

B. FSM Mapping

In this step channels are mapped to the FSM that was created during the analysis of the protocol. First, the state machine is split into a master FSM and a slave FSM assigning them the transaction phases that belong to each side. Second, both the master and slave FSMs have to be split into multiple state machines, each dedicated to the processing of a different channel. When doing so, care should be taken that inter-channel dependencies are properly synchronized among state machines.

For the DTL protocol each channel has its own FSM in both the master and slave transactors. Each of the DTL channels has two phases and one phase of every channel is on the master side and the other one in the slave side. This means that there are a total of eight state machines for the entire protocol. Figure 3 shows the DTL channels and the phases that are associated with each channel. The figure also shows the phase flows that can occur in a DTL transaction. The FSM for each phase processes the protocol attributes (payload extensions) that are associated with that phase. The state machines are designed to enable pipelining of phases. There are also some inter-channel phase dependencies (as can be seen in Figure 3), which means that synchronization between the FSMs of the different channels is needed.

VI. Transactor Creation

The final step is the creation of the master and slave transactors that are needed to connect the core IP models with generic SCML interfaces to the protocol-specific TLM 2.0 interface. The basic structure of the transactors is always the same, therefore a set of transactor templates have been defined to simplify their modeling.

A. Transactor Templates
Both the master and slave transactors have an active part that initiates outgoing transactions and a reactive part that processes the incoming transactions. The TLM 2.0 socket interfaces contain both a forward and a backward path. The forward path is used to send transactions from the master to the slave, while the backward path is used to send transactions from the slave to the master. Therefore, the active part of the master transactor drives the forward path and the active part of the slave transactor drives the backward path. The reactive parts of both the master and slave transactors implement the nb_transport_(fw/bw) interface method and handles incoming transport calls. Between the forward and backward paths of the transactors a synchronization layer is required. Finally, both transactors have a generic high-level interface that handles the communication on the IP side.

B. DTL transactors example

The task of implementing the transactors for a specific protocol consists of taking the sockets and protocol state machines that were created in the TLM 2.0 mapping step and filling in the transactor templates. The active parts of the transactor templates are filled in with one or more FSM or thread processes. Mechanisms for the synchronization between the different processes and between the active and reactive parts within the transactors need to be implemented, for example with events for control (phase) synchronization and FIFOs for data synchronization.

Figure 4 shows the block diagram of the master transactor for the DTL protocol. Synchronization between the state machines, threads and generic interfaces is realized with default SystemC events and FIFOs. The nb transport bw function that implements the TLM 2.0 backward path in the master transactor notifies the correct FSM whenever a particular phase was received. For a more detailed description of the DTL transactors, please refer to [12]

VII. RESULTS

Performance of the CA DTL interface and transactors has been tested in a set-up with a simple master (traffic generator) and slave (memory) connected directly together.

The simulation speed of the testbench system in clock-cycles per second (CPS) is calculated as follows:

\[ CPS = \frac{t_{sim}}{t_{CPU} \times d_{cycle}}, \]

where \( t_{sim} \) is the elapsed simulation time, \( t_{CPU} \) is the physical time it took to simulate the testbench and \( d_{cycle} \) is the reference clock period. Figure 5 shows the speed measurement results. The traffic generator issues 2 million alternating read/write transactions. The test is done for burst sizes of 1, 4, 8, 16, 32 and 64 words. The testbench is simulated with two different configurations. Configuration 1 uses no delays and every clock-cycle a data item is transferred. Configuration 2 simulates processing delays by introducing a 2 cycle delay for every command and data phase.

As shown in Figure 5, the CPS increases with bigger burst sizes. This is because although both \( t_{sim} \) and \( t_{CPU} \) increase with bigger burst sizes, they do not increase at the same rate. Furthermore, it can be clearly seen that while the introduction of computational (processing) delays in configuration 2 increases the simulation time, the CPU time stays about the same. This means that the ratio cycles/second improves when delays are added to the simulation. The reason for this is that the transactors do not use clock sensitive processes that are evaluated every clock-cycle (as in signal-level models), but processes are awake only when there is a transaction taking place.

VIII. CONCLUSIONS AND FUTURE WORK

A methodology to create protocol specific BCA interfaces and transactors using the TLM 2.0 standard is presented in this paper. This work demonstrates that: 1) the extension mechanisms of the TLM 2.0 standard can be used to create a protocol specific BCA interface and 2) the simulation speed of the resulting models is between one and two orders of magnitude better than signal-level RTL models (maximum speed of 700 kcycles/second), while retaining the same level of accuracy.

REFERENCES