EFFICIENT AND EXTENSIBLE TRANSACTION LEVEL MODELING BASED ON AN OBJECT ORIENTED MODEL OF BUS TRANSACTIONS

Rauf Salimi Khaligh, Martin Radetzki
Institut für Technische Informatik,Universität Stuttgart
Pfaffenwaldring 47
D-70569 Stuttgart, Germany
rauf.salimi@informatik.uni-stuttgart.de
martin.radetzki@informatik.uni-stuttgart.de

Abstract

Transaction level modeling (TLM) aims at abstract description and high-performance simulation of bus-based System-on-Chip (SoC) platforms. While languages and libraries providing the necessary basic modeling mechanisms are available, there is ongoing discussion on modeling styles for their usage. This contribution employs object-oriented modeling of transaction data and bus protocols, targeting at better extensibility and reuse of models. Our simulation performance measurements show that by abstracting from signal-level data representations, a cycle-accurate object-oriented model can achieve performance close to models on cycle-approximate level.

Keywords: Transaction-Level Modeling, SystemC, Object-Oriented, Embedded Systems

1. Introduction

Embedded systems and systems on chip integrate an increasing number of processor nodes and complex communication structures. It is an essential design task to model these systems prior to their implementation. This enables an early validation of system architecture concepts, exploration of architectural alternatives, and performance evaluation. Modeling on register transfer level (RTL) is no longer an option due to the complexity, inflexibility, and low simulation performance of the resulting models. Transaction level modeling has been devised as an alternative.

One key principle of TLM is to perform communication by function calls rather than via signal communication. The other is to exchange information on a coarser level of granularity compared to low-level signals. With these constraints, bus based systems can be described at different TLM abstraction levels. The abstraction levels representing different levels of accuracy required in most modeling activities have been identified and proposed by some researchers and institutes active in the transaction level modeling field (e.g. [6, 5, 12]). For example, in The most abstract level, the so-called functional view (FV) abstracts completely from the communication architecture by modeling point-to-point connections between each pair of communication partners. In the programmer's view (PV), bus communication is modelled under abstraction from timing and arbitration. The architecture view (AV), also called cycle approximate (CX) view, adds approximated arbitration and timing. In the cycle accurate (CA) or verification view (VV), transactions must allow to reproduce for each cycle the bus owner and transaction that is active in an RT level implementation of the bus.

The SystemC TLM standard defines interfaces for initiating and receiving transactions as well as basic TLM fifo channels. It does not describe a systematic modeling style for the transactions themselves. The contribution of this work is an object-oriented modeling of transactions that leads to easily extensible cycle-accurate models which approach the simulation speed of less precise CX models.

The remainder of this paper is organized as follows. In Section 2, we summarize related work. Section 3 provides a summary of our object-oriented transaction models. In Section 4, we present its application to cycle-accurate modeling of the AMBA AHB. Section 5 presents experimental results on the achievable simulation performance, and Section 6 concludes this paper.

2. Related Work

The essence of transaction level modeling (TLM) is language and application domain independent. Examples of modeling languages explicitly enabling transaction level modeling are SystemC [8] and SpecC [7]. In the relatively young field of TLM, communication schemes, interoperability, definition of abstraction levels and approximations and the terminology are constant subjects of debate and standardization effort. Examples are works of the Open SystemC Initiative (OSCI), the Open Core Protocol International Partnership (OCP-IP) and the SpecC modeling community [3, 6, 5, 12].

In its current version (1.0), the OSCI-TLM standard [12] proposes use of certain interfaces and channels based on these interfaces. The OSCI-TLM standard addresses several issues such as separation of user code from the communication code in layers, module interoperability and reuse. However, the modeling style presented in the OSCI-TLM white paper has a number of drawbacks. Modeling passive components deviates significantly from modeling active components based on the presented modeling style. The transition from passive to active models is often required when moving from higher to lower levels of abstraction. Another drawback is use of C *structs* to transfer information. This affects the extensibility and reuse in the models. These issues have been addressed in [11] and a new modeling style – the Object Oriented Transaction Level Modeling (OOTLM) approach – was proposed and demonstrated using a simple model. However, the applicability of this method to complex real-world communication protocols was not proven and its performance implications and accuracy limitations were not studied in detail. In this paper a protocol widely used in embedded systems, the AMBA AHB [1] is modelled based on the OOTLM approach to investigate its applicability and the achievable simulation performance.

A recent work in the OSCI-TLM domain is GreenBus [9] which provides a generic, protocol neutral interconnection scheme and library for SystemC models on top of the OSCI-TLM standard which can be customized to model concrete bus protocols. In GreenBus, a bus transaction is composed of uninterruptible transfers called *atoms* which transfer basic data elements referred to as *quarks*. Granularity of the information being monitored or updated by components connected to a GreenBus based bus depends on the abstraction level of the model.

The AMBA AHB has been used by many researchers in evaluation of their modeling approaches. In [10] Pasricha et al have developed models of the AHB based on their proposed CCATB abstraction level. In [13] Schirner et al compare accuracy and simulation performance of their SpecC based AHB models in different levels of abstraction. Caldari et al [4] have developed transaction level models of the AHB in SystemC 2.0. Their work is not based on the OSCI-TLM standard. Recently the ARM specific Cycle Accurate Simulation Interface (CASI) and CASI based AMBA models have been released by ARM [2]. CASI models are essentially cycle-based models, and to achieve high simulation speeds avoid using SystemC events (except for clock events).

3. The OOTLM Approach

In OOTLM, the interactions between model elements are represented by transaction objects. A transaction object represents read or write transfers, reset or initialize operations or abstractions of more complex interactions and encodes the information required for the particular interaction. For example, for a reset operation, this would be the address of the target to be reset. For a read transfer, attributes of the transaction object represent the address of the data item to be transfered and the data to be returned by the target.

Transaction objects are created and initialized by the initiating masters and are sent to the bus model via OSCI-TLM compliant channels. Subject to the arbitration policy and algorithm, the transaction objects are eventually routed

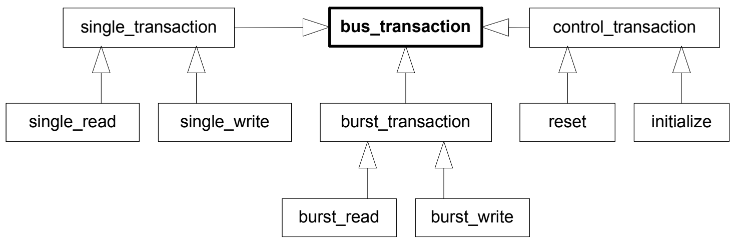

Figure 1. A generic transaction class hierarchy

to the addressed slaves to be *processed*. For example, for read transfers, the slave updates the transaction object with the correct response and data values. During the life time of a transaction object, the initiating master has access to it through a pointer and can query or change its attributes. This way, without loss of precision, the additional data traffic and the overhead of using separate request, response and control transactions is avoided. A generic example of transaction classes that could be used to model the operations in a typical bus is shown in the UML class diagram of figure 1. Using an appropriately designed transaction class hierarchy, a single OSCI-TLM based channel can be used to transport different types of transaction objects (e.g. with `put(T &t)` and `get(T &t)` methods). In this example this would be a channel of type `tlm_fifo<bus_transaction*>` and would result in the simple interconnection scheme shown in figure 2.

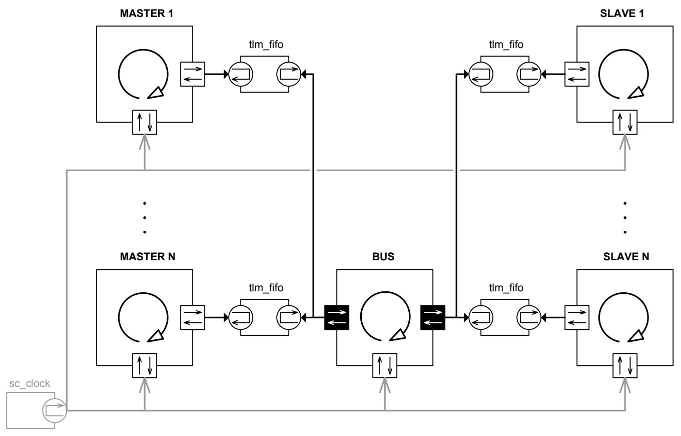

Figure 2. Model interconnection

As a modeling technique enabling reuse of passive slave models (see [11]), interaction of the slaves with the transaction objects is performed using the existing methods of the slaves (e.g. `read()` and `write()`). The next sections also show how the OOTLM approach enables reuse, incremental development and extensibility of the models.

4. A Cycle Accurate OOTLM Model of the AMBA AHB

The Advanced High Performance Bus (AHB) is part of the AMBA family of busses from ARM [1] and is intended to be used as a backbone bus in systems-on-a-chip (SoCs). The AHB is a multi-master, pipelined bus featuring single cycle bus handover, fixed size bursts, incremental bursts of unspecified length and split and retry transfers.

In this work, we have focused on a subset of the AHB specification large enough to validate the OOTLM approach in modeling complex bus protocols. We have modelled single data item transfers and fixed-length bursts. From the possible slave responses, we have chosen `OKAY`, `ERROR` and `SPLIT` responses. As a simplifying assumption, preemption of unlocked bursts has not been modelled and a burst can only be terminated prior to its completion as a result of a `SPLIT` response. However as mentioned in section 4.1, this does not affect the generality of our approach and the developed models can be modified to support unlocked bursts in a similar fashion to split transfers.

The developed model is cycle accurate. The arbitration, bus handover, inter and intra-transaction timing and status of the transfers in each cycle are accurate and fully compliant with the AHB specification.
The basis of our model is the pipelined nature of the AHB, shown in an abstract form in figure 3. AHB transfers have two phases, the *address phase* and the *data phase*. In each bus cycle, two different transfers may be underway concurrently on the bus. One in the address phase and one in the data phase. A transfer in the address phase progresses to the data phase, when the transfer currently in the data phase completes.

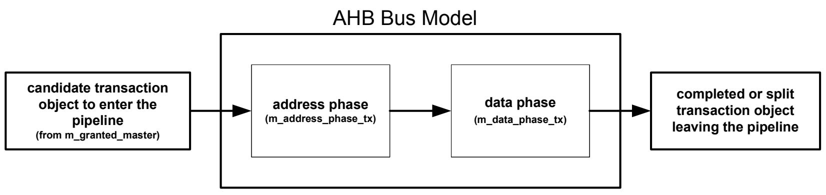
AHB Bus Model
candidate transaction object to enter the pipeline
(from m_granted_master)
address phase
(m_address_phase_tx)
data phase
(m_data_phase_tx)
completed or split transaction object leaving the pipeline

Figure 3. An abstract view of the AHB bus as a pipeline

Within the bus model (section 4.2), this abstract pipelined view of the AHB is maintained and updated in each cycle using two pointers, one to the transaction in the address phase (`m_address_phase_tx`) and one to the transaction in the data phase (`m_data_phase_tx`). Depending on status of `m_data_phase_tx`, the bus model coordinates routing of the transaction objects and other timing aspects of the AHB bus.

4.1 AHB Data Transfer Transactions

The transaction objects are the means of information transfer between masters and slaves and can be considered virtual communication paths created dynamically by the bus model. Figure 4 shows the transaction class hierarchy modeling the AHB single transfers and bursts. Some attributes represent the address, data and control information and have direct counterparts in the AHB signal set (e.g. `m_transfer`). Other attributes are used to model the dynamics of the transaction objects, delayed data bus handover and split transfers. The slave response and ready status are visible to the initiating master and can be polled. Alternatively, a master can wait for the completion of transaction without polling by waiting on `m_completion_event`.

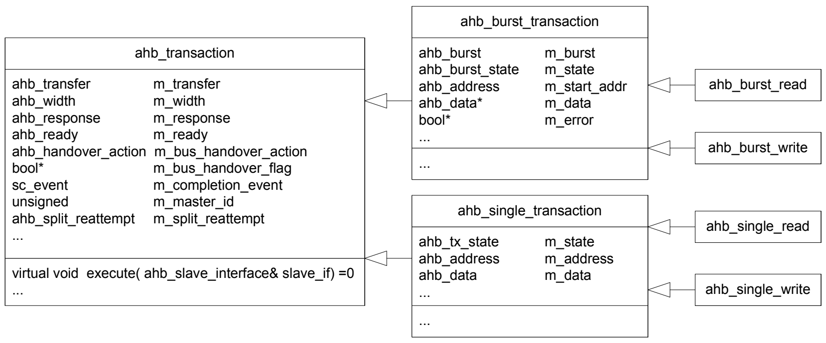

Figure 4. AHB transaction class hierarchy

The slave processing a transaction object calls the `execute()` method, passing a pointer to its `ahb_slave_interface` (figure 6), to update the attributes of the transaction object. To get the data, response and ready status, the `execute()` method will in turn call the `read()` and `write()` interface methods of the target slave whenever necessary.

Based on the properties of the bursts and split transfers, the amount of object traffic between model elements is reduced to improve performance. For split transfers, the bus and slave models keep pointers to the split transaction objects in a list of *pending* transactions. The slave signals to the arbitration algorithm its readiness to process the transaction using the `m_split_reattempt` attribute of the transaction object. Similarly, the re-attempting master does not need to send a new transaction object to the slave, the bus model re-enables the transaction object to be re-processed by the slave. AHB bursts are not allowed to cross slave address boundaries. Thus, for every burst a single transaction object is sent to the addressed slave.

Although not modeled and implemented in this work, preemption of unlocked bursts (as the result of the master loosing the bus) can also be modeled similarly with the aforementioned *pending* transactions, with simple modifications to the model. The initiating master of a burst can not perform any other transactions until all beats of the burst are completed either successfully or with an error. Again, a single burst transaction object needs to be sent to the slave by the master. If the master loses the bus prematurely, the burst transaction will become a pending transaction, waiting to be re-enabled by the bus model when the master takes ownership of the bus.

The dynamic behavior of the transaction objects is modelled by state machines. The state machine for `ahb_single_transaction` is shown in figure 5. Burst transactions are modelled with a concurrent state machine, observing the fact that a burst can have concurrent beats, one in the address phase and one in the data phase. The burst state machine is omitted here to save space.

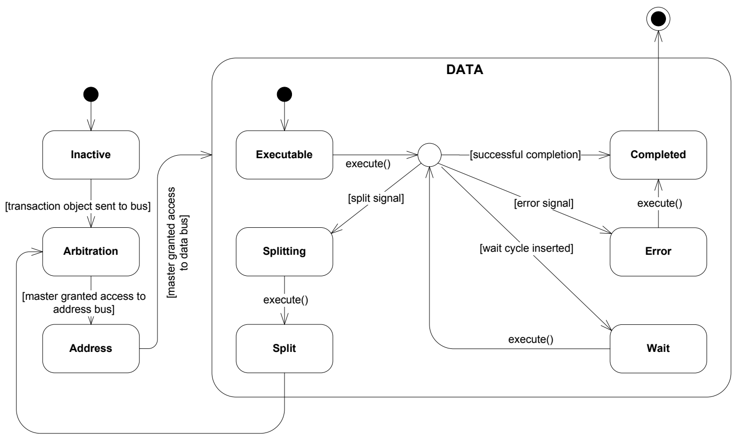

Figure 5. Single transaction state machine

4.2 The AHB Bus Model

The bus model is responsible for routing of the transaction objects to their addressed slaves and is partially shown in figure 6. The arbitration algorithm is implemented in the `select_master()` method and considers transactions in the master-bus channels and the pending transaction list in selecting the highest priority master. An entry in the pending transaction list corresponds to the request of a master which has received a split transfer and is waiting to be granted the bus for a re-attempt. A data structure mapping address ranges to slaves is used for decoding, and the transaction objects are routed to the addressed slaves either via the slave-bus channel (for new transactions) or by putting the transaction objects in the address phase again (for split transactions, figure 5).

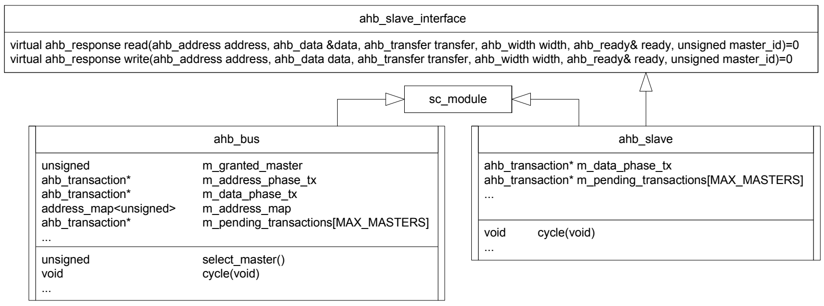
ahb_slave_interface
virtual ahb_response read(ahb_address address, ahb_data &data, ahb_transfer transfer, ahb_width width, ahb_ready& ready, unsigned master_id)=0
virtual ahb_response write(ahb_address address, ahb_data data, ahb_transfer transfer, ahb_width width, ahb_ready& ready, unsigned master_id)=0
sc_module
ahb_bus
unsigned m_granted_master
ahb_transaction* m_address_phase_tx
ahb_transaction* m_data_phase_tx
address_map<unsigned> m_address_map
ahb_transaction* m_pending_transactions[MAX_MASTERS]
...
unsigned select_master()
void cycle(void)
...
ahb_slave
ahb_transaction* m_data_phase_tx
ahb_transaction* m_pending_transactions[MAX_MASTERS]
...
void cycle(void)
...

Figure 6. Bus and slave models

The functionality of the bus is implemented in the clock triggered process `cycle()` and is modelled by a state machine (figure 7). Based on the pipeline model of the bus (figure 3), arbitration and all aspects of the AHB bus are modelled accurately using this state machine.

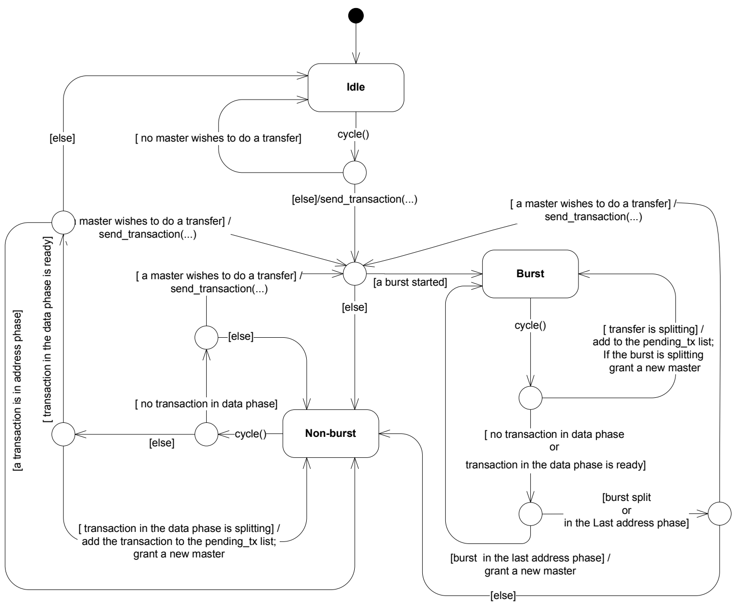
Idle
[else]
[no master wishes to do a transfer]
cycle()
[else]/send_transaction(...)
[a master wishes to do a transfer] / send_transaction(...)
master wishes to do a transfer] / send_transaction(...)
[a master wishes to do a transfer] / send_transaction(...)
[a burst started]
Burst
[else]
[else]
cycle()
[transfer is splitting] / add to the pending_tx list; If the burst is splitting grant a new master
[no transaction in data phase]
Non-burst
cycle()
[else]
[no transaction in data phase or transaction in the data phase is ready]
[burst split or in the Last address phase]
[a transaction is in address phase]
[transaction in the data phase is ready]
[transaction in the data phase is splitting] / add the transaction to the pending_tx list; grant a new master
[burst in the last address phase] / grant a new master
[else]

Figure 7. Bus state machine

Together with the bus model, we also developed generic master and slave models for performance measurement and validation purposes. Because of the inheritance relationship between the transaction classes and since the models were designed based on a set of polymorphic methods of the transaction classes, we were able to develop the models incrementally and in three steps, extending the models by subclassing. In the first step, the basic elements of the AHB functionality – arbitration, decoding/routing, pipelined behavior and single transactions – were modelled and implemented. In the second step split transactions, and in the last step bursts and burst related issues were implemented. For example, to extend the bus model from the first step to support split transactions, one new transition was required in the bus state machine, and the model was extended by adding attributes, one new event and overriding some methods. As another example, no changes were necessary in the slave from the second step to enable processing of burst transactions.

5. Experimental Results

We implemented the models using the OSCI SystemC library version 2.1 v1 with Microsoft Visual C++ .NET 2003 in Microsoft Windows XP professional. The bus model was validated against the protocol specification using deterministic and randomized tests. To measure the simulation performance of the bus model, we used a single master, single slave test setup similar to [13] and measured performance indices also reported in other related work [9, 4].

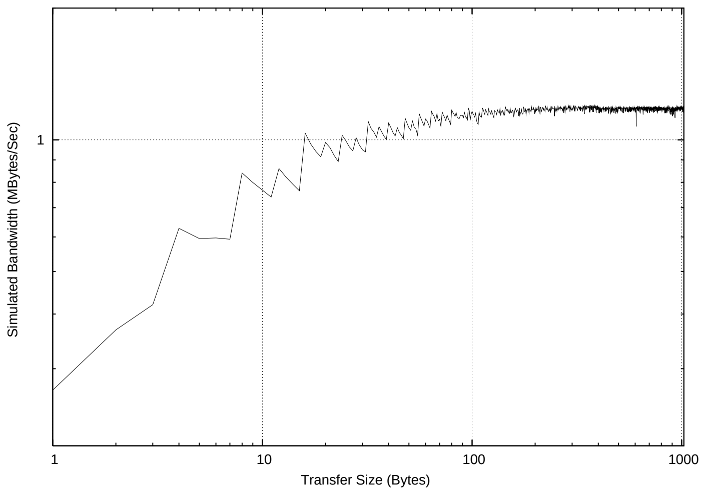

Figure 8. Simulated bandwidth

First, we measured the simulated transaction processing time and equivalently the simulated bandwidth. These indices represent the minimum amount of (real or wall clock) time required to perform a data transfer transaction of a certain size using the most efficient combination of bursts and single transfers. Figure 8 shows the measured simulated bandwidth for transactions ranging in size from 1 byte to 1024 bytes. Each transaction was performed 1000 times (all with OKAY responses and no wait cycles), and the average time was recorded. The measurements were performed on a workstation with a 2.21 GHz AMD Athlon(tm) 64 3500+ processor with 512 kb of Cache and 1 GB of RAM.

The saw-tooth shape of the graph is the result of breaking the transfers into a combination of bursts and single transfers by the master. For small transfers, the overhead of transfer of transaction objects is relatively high and the worst case bandwidth reaches a value of 0.270 MBytes/Sec. As the transfer size increases, this overhead becomes negligible and the bandwidth approaches a value of 1.2 MBytes/Sec.

In a similar test setup but in a different simulation environment, the bus functional model (BFM) of the SpecC-based model of Schirner and Dömer [13] reaches a simulated bandwidth of 0.03 MBytes/Sec and their next most accurate model (ATLM, which is essentially a cycle-approximate model), reaches 2.29 MBytes/Sec. In [9] performance figures of bus accurate GreenBus based models of the IBM PLB are presented, which expressed in terms of simulated bandwidth, reach a maximum of 2.8 MBytes/Sec.

It should be noted that our reported simulated bandwidth values are very conservative. We have used a single data width for all transfers (e.g. byte). For example, in our test setup, to transfer three bytes, three bus transactions are used, each transferring a single byte. In the ATLM model in [13] on the other hand, three bytes are transferred in two transactions, one transferring a byte and one transferring an AHB half-word (two bytes). In our model, transferring three individual words (each word being four bytes) would take the same amount of time required to transfer three individual bytes. Considering this, the simulated bandwidth of our model approaches 4.8 MBytes/Sec. However, we have decided to report the more conservative values, as the aforementioned details regarding transfer setup were not explained in some published related work (e.g. [9]).

Next, we measured the model simulation speed which is expressed in terms of simulated number of clock cycles of a simulation model per unit of time. The average model simulation speed for our test setup reached 1150 KCycles/Sec. Caldari et al [4] have reported a simulation speed of 300 KCycles/Sec for their SystemC 2.0 based models. They have used a more complex master model and have performed their measurements on a Sun Ultra 60 Workstation.

6. Conclusions

We have shown that, based on an abstract object-oriented modeling style, easily extensible and cycle accurate modeling of a complex, real world bus system and protocol can be performed. Instead of using discrete transactions for arbitration, control, address and data, we have modelled bus transactions using complex transaction objects which encode accurately all the necessary information, and whose dynamic behavior represents different phases of the bus transfers accurately down to the individual buy cycles. This results in reduced traffic between model elements which in turn leads to higher simulation performance, without loss of precision. Our measurements, in comparison to the related work, show a simulation performance that is significantly above other cycle-accurate models and comes close to the performance of models that are cycle-approximate. By further reducing the amount of SystemC events and by moving to the cycle-approximate level, we plan to achieve a further speed-up. Future work includes the application to additional bus protocols and the development of a simulation library of master and slave components as well as a connection of our model to instruction set simulators.

References

[1] ARM Limited. *AMBA Specification, version 2.0*, May 1999.

[2] ARM Limited. *AMBA AHB transaction level modeling specification, version 1.0.1*, May 2006.

[3] L. Cai and D. Gajski. Transaction level modeling: an overview. In *Proceedings of CODES+ISSS '03*. ACM Press, 2003.

[4] M. Caldari, M. Conti, M. Coppola, S. Curaba, L. Pieralisi, and C. Turchetti. Transaction-level models for AMBA bus architecture using SystemC 2.0. In *Proceeding of DATE '03*. IEEE Computer Society, 2003.

[5] J. A. Colgan and P. Hardee. *Advancing Transaction Level Modeling, Whitepaper*. CoWare, Inc., 2004.

[6] A. Donlin. Transaction level modeling: flows and use models. In *Proceedings of CODES+ISSS '04*. ACM Press, 2004.

[7] R. Dömer, A. Gerstlauer, and D. Gajski. *The SpecC Language Reference Manual, version 2.0*. SpecC Technology Open Consortium, Dec 2002.

[8] IEEE Computer Society. *IEEE Standard SystemC language reference manual,IEEE Standard 1666-2005*, 2005.

[9] W. Klingauf, R. Günzel, O. Bringmann, P. Parfuntseu, and M. Burton. Greenbus: a generic interconnect fabric for transaction level modelling. In *Proceedings of DAC '06*. ACM Press, 2006.

[10] S. Pasricha, N. Dutt, and M. Ben-Romdhane. Fast exploration of bus-based on-chip communication architectures. In *Proceedings of CODES+ISSS '04*. ACM Press, 2004.

[11] M. Radetzki. SystemC TLM Transaction Modelling and Dispatch for Active Objects. In *Proceedings of FDL'06*, Darmstadt, Germany, 2006.

[12] A. Rose, S. Swan, J. Pierce, and J.-M. Fernandez. *Transaction level modeling in SystemC, Whitepaper*. Open SystemC Initiative, 2004.

[13] G. Schirner and R. Dömer. Quantitative analysis of transaction level models for the AMBA bus. In *Proceedings of DATE '06*. European Design and Automation Association, 2006.