Modelling Alternatives for Cycle Approximate Bus TLMs

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

Abstract
Transaction level models (TLMs) can be constructed at different levels of abstraction, denoted as untimed (UT), cycle-approximate (CX), and cycle accurate (CA) in this paper. The choice of a level has an impact on simulation accuracy and performance and makes a level suitable for specific use cases, e.g. virtual prototyping, architectural exploration, and verification. Whereas the untimed and cycle-accurate levels have a relatively precise definition, cycle-approximate spans a wide space of modelling alternatives between UT and CA, which makes it a class of levels rather than a single level. In this contribution we review these modelling alternatives in the context of SystemC and with focus on bus models, provide quantitative measurements on major alternatives, and propose a CX modelling level that allows to obtain almost cycle accuracy and a simulation performance significantly above CA models.

1. Introduction

Transaction level modelling has become a widely used technique in embedded systems and system on chip design. A variety of system design languages such as SystemC [7] and SpecC [5] can be used for modelling at transaction level. However, transactions and many other typical elements of transaction level models (TLMs) are not available as syntactic language features. The TLM creators instead have to create the transaction level abstractions themselves, using language features such as channels and interfaces. This is supported by mostly informal descriptions of the TLM methodology, e.g. [6], and by methodology-specific libraries, e.g. the SystemC TLM library [10].

Methodologies and libraries leave degrees of freedom to implement TLMs in different ways. This has the positive effect that the transaction level in fact spans multiple (sub-)levels of abstraction, facilitating trade-offs between simulation accuracy and performance. However, these levels, subsequently denoted as untimed (UT), cycle-approximate (CX) and cycle accurate (CA) are not formally defined but rather characterized by model properties. The lack of a formal definition makes it difficult to describe how to systematically construct TLMs at a given level.

Despite this drawback, there exist relatively precise and consistent characterizations of UT and CA, as we will show in section 2. CX models, however, can cover a wide range between UT and CA, and there appears to be no consensus on the characteristics of a favourable CX model. We will attempt the definition of such a model based on the consideration of modelling alternatives. For this purpose, we use the following non-orthogonal criteria characterizing TLMs in addition to their timing accuracy:

- The underlying communication mechanism, which can be a subprogram call with transfer of control flow (blocking) or message passing with data flow (potentially non-blocking).
- The use of concurrency in the model, namely the presence or absence of individual threads in the modelled master, slave, and bus components. A component with (without) a thread is called passive (active).
- The programming abstraction provided to the users of a bus model, including no abstraction (direct access to port/channel), procedural application programming interface (API), communication mechanisms that could be adopted from concurrent / distributed systems (e.g. RPC, CORBA).
- The bus features covered by the model, including single transfers, bursts, locked transfers, split transfers, wait states (inserted by slave), busy cycles (inserted by master), bus phases and pipelining, in-order or out-of-order completion of transfers, and arbitration policy.
- The modelling mechanism used for arbitration, in particular the use of events to trigger arbitration (no events, one event, multiple events).
- The use cases of a particular model, including verification, exploration, virtual prototyping.

In the next section, we review the related work with respect to the above criteria. Section 3 presents considerations and alternatives towards accurate CX models, and section 4 investigates their performance.
2. Related Work

Donlin [4] presents the transaction level terminology used by the SystemC TLM working group. It includes a Programmer’s View (PV) characterized by untimed communication and the use case of providing a functionally accurate representation of hardware subsystems to software programmers. A Programmer’s View with Time (PV+T) results from annotating a PV model with time and approximate arbitration. A Cycle Accurate (CA) view is characterized by fully bus protocol compliant arbitration and timing accurate to the level of individual cycles.

In the OCP terminology [9], three TLM layers are defined: The Transfer Layer (L-1) is characterized by cycle-true behaviour and use for verification and precise simulation. At the Transaction Layer (L-2), modelling abstracts from the details of a bus protocol but can take properties like split transactions and pipelining into account. The Messaging Layer (L-3) is untimed and enables 1:1 connections between initiators and targets, abstracting from bus address mapping.

The SpecC related taxonomy from [3] takes into account the timing accuracy of computation as orthogonal to the communication timing aspect and defines cycle-timed, approximate-timed and untimed levels for both dimensions. Considering the communication dimension only and focusing on TLM models, we can identify an untimed component-assembly model (CAM) which models communication between system components by message passing, a bus arbitration model (BAM) with arbitration policy modelling that approximates timing by one wait statement per transaction, and a cycle-timed bus-functional model (BFM).

The GreenBus approach [8] makes a significant step towards a constructive definition of transaction levels. It identifies three levels of granularity called transactions, atoms, and quarks. A transaction is a sequence of uninterruptible phases (atoms), and each atom is a collection of payload values (quarks). A PV model approximates timing at transaction boundaries, a bus accurate (BA) model at atom boundaries, and a cycle callable (CC) model must model all quark updates with cycle accuracy. An untimed model is not defined.

From these considerations, it is apparent that there still exists no unified terminology in the TLM field. Table 1 classifies the modelling levels described in the aforementioned approaches with respect to their bus communication timing properties.

<table>
<thead>
<tr>
<th>Comm. timing</th>
<th>UT</th>
<th>CX</th>
<th>CA</th>
</tr>
</thead>
<tbody>
<tr>
<td>SystemC TLM</td>
<td>PV</td>
<td>PV+T</td>
<td>CA</td>
</tr>
<tr>
<td>OCP L-3</td>
<td>L-2</td>
<td>L-1</td>
<td></td>
</tr>
<tr>
<td>Cai / SpecC</td>
<td>CAM</td>
<td>BAM</td>
<td>BFM</td>
</tr>
<tr>
<td>GreenBus</td>
<td>PV(+T), BA</td>
<td>CC</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Overview of Transaction Levels

The UT approaches have in common the primary use case of virtual system prototyping and that they result in a purely functional simulation. This limits the available choices with respect to our characterization criteria as well as the impact of the remaining choices on the simulation result. Subtle differences exist—for example, the SpecC approach features message passing and active slaves at the CAM level whereas SystemC PV uses function calls from masters into passive slaves—but these should not have impact on the functional result of simulation nor the non-existent timing (whereas an impact on simulation performance is likely). Another such difference is whether bus structure, addressing scheme, and approximate arbitration are modelled (SystemC PV) or not (point-to-point connections in OCP L-3).

A similar situation can be observed at the CA level. The primary use cases are verification reference and precise performance analysis. The property of cycle accuracy strongly restricts the modelling space. All bus features must be modelled, communication is necessarily by non-blocking data flow between concurrent components, and arbitration is typically performed in each cycle. A detailed investigation of CA model code often reveals that some interface abstraction is provided, but “under the hood” the model implements communication at the level of the signals used in the bus protocol, even if these are bundled in a TLM channel. For example, Table 1 in [8] shows the direct correspondence between GreenBus quarks and protocol signals. In the SystemC based AMBA cycle accurate simulation interface (CASI) [2], the CA AHB channel uses a data structure whose attributes are identical to the AHB signals. A proposal for more abstract protocol modelling based on hierarchical state machines has been made in [13].

At the CX level with the primary use case of system exploration and performance (bus throughput or latency) estimation, a much wider range of modelling alternatives exist. Within the SystemC TLM and GreenBus PV+T models, timing is approximated at the granularity of transactions, arbitration abstracts from the precise
bus arbitration policy, and transactions cannot be pre-empted. Thus, features such as split transactions cannot be modelled. On the other hand, the SpecC BAM and GreenBus BA models permit pre-emption of transactions and subsequent bus re-arbitration. Thereby, more precise simulation can be obtained at the cost of lower simulation performance compared to PV+T.

An interesting approach to CX modelling is presented in [14], where transactions are simulated with the optimistic assumption of not being pre-empted. If this assumption turns out to false at a later simulation time, the transaction duration is extended by the duration of pre-empting transactions. This yields a 100% accurate simulation with respect to the author’s measure of timing accuracy. However, the data of a burst transaction are transferred in a single operation at the beginning of that transaction. This means that individual data transfers are not cycle accurate and the interleaving of data from pre-empting transactions cannot be simulated, which may affect data-dependent functionality.

In the remainder of this paper, we investigate whether a CX model can be designed to cover a maximum of bus features and to come as close as possible to cycle accuracy, including accuracy of the data transfers. We will also investigate modelling decisions that optimize simulation performance without impacting accuracy. The resulting model can provide rather accurate estimates for the purpose of system exploration, complementing the significantly less accurate yet faster PV+T models.

3. Modelling Decisions

Since we target a SystemC model implementation, we will use the SystemC TLM terminology in the following but keep the term CX for our model.

3.1. Concurrency

In most PV and PV+T models, slaves are passive and masters are active components. This limits the achievable accuracy because master and slave cannot operate concurrently. For example, a master cannot prepare data for the next transaction while a slave processes the master’s current transaction request. To avoid this possible deviation from detailed system timing, we choose to make slaves active components in our CX model.

Another modelling alternative pertains to the modelling of the bus as an active or passive component. This is closely related to arbitration modelling, discussed in section 3.5.

3.2. Communication Mechanisms

PV and PV+T models typically employ transfer of control flow (blocking subprogram calls) as a mechanism for communication between master and slave. This is in conflict with the desired concurrency of master and slaves. Therefore, we use data flow to pass messages between communicating blocks. However, for large message payloads such as burst data, we use a shared memory implementation where only a pointer to the shared data is passed as part of the message. Thereby, copying of the payload is avoided and simulation performance increased. Access conflicts on the shared memory are avoided by limiting access by the communication partners (master, slave, bus model) to disjoint phases of the transaction. The dynamic memory management is handled by the master’s port, hidden from the user, to avoid memory leaks and dangling pointers. Memory is allocated upon start of a transaction and freed when the master has obtained the last data of a transaction response according to the programming model (cf. section 3.3).

We have tried to avoid a suspected overhead due to repeated creation and deletion of memory blocks by reusing a pool of such blocks. This had no significant impact on simulation speed; possibly because such optimization is already implemented in the C++ runtime library’s heap management.

Another modelling choice must be made between use of standard TLM channels (tlm_fifo) to connect masters and slaves with the bus model vs. direct connection to interfaces exported by the bus, cf. Figure 1. The latter option is likely to be more efficient because it avoids the overhead of storing and retrieving messages in/from a tlm_fifo. Moreover, the master’s interface method calls will go directly into the bus model, enabling an implementation that reduces the number of context switches during simulation. Both variants have been implemented and their resulting simulation performance is compared in section 4.

![Figure 1: Alternative Communication Modelling](image-url)
3.3. Programming Abstraction

In most PV and PV+T models, subprogram calls serve as a well-understood programming abstraction of communication operations. However, subprogram calls cannot be used between concurrent or distributed model objects. Remote procedure calls (RPC) are a mechanism that could be adapted from distributed programming; however, in the presence of return parameters, they would block the master which would compromise the accuracy of our model. Mechanisms like CORBA enable non-blocking communication, but they are too heavyweight for use in a fast transaction level simulation.

As a compromise, we have adapted the active object design pattern known from concurrent object-oriented programming for the purpose of TLM, see Figure 2. A key concept of this pattern is the future object which is immediately returned as result of a non-blocking subprogram call. In our adaptation, the call models a non-blocking communication operation and the future object can later be used by the master to obtain results from that operation (the transaction response) at its own discretion. Beyond that, the future object is also used to allow the master to delay the supply of values that belong to the transaction request to a time after request initiation. Thereby we can accurately model that the master may supply (retrieve) bus word number \( i \) up to (starting from) the \( i \)-th cycle after the start of a burst transfer instead of providing or receiving all burst data at the beginning or at the end of a transaction.

![Figure 2: Active object pattern for TLM](image)

Another active object concept is the guard which can be defined individually per operation at the server (slave) side, cf. Figure 2. We utilize the guard as a programming abstraction of a bus feature that allows a slave to split a transaction that it cannot serve, and to resume that transaction when appropriate. As an efficiency improvement over [12], we have modelled an event-based resume mechanism to avoid polling the guard in each cycle during which a transaction is split. Moreover, the models presented in this paper facilitate for the first time the splitting of burst transaction at the granularity of the transaction rather than single bus word transfers.

3.4. Bus Features

The most basic bus features are single bus word read and write transactions (single transfers). Successive transfers to consecutive addresses can be combined into a burst transaction. Burst transactions may have a fixed or user-defined length. They may be preemptible or not (locked). The burst address sequence may wrap around at block boundaries (wrapping burst) or not. We model all these transactions and transaction properties in an object-oriented way as C++ classes and attributes (data members). Details about this modelling style can be found in [11].

Another feature found in most high performance buses is pipelining. To employ pipelining, transactions are decomposed into phases, and different phases of subsequent transactions are allowed to execute in parallel. We model the phase as a state attribute of a transaction which is controlled by the bus model. Pipelining can be modelled in a cycle accurate way by introducing a number of stages into the bus model as shown in [13]. Our CX model covers pipelining within a single transaction (which is relevant for burst transactions), but neglects it at the boundary between different transactions for performance reasons.

We model split transactions using the guard mechanism for abstraction as presented in the previous subsection. The OCP L-2 model is the only other CX model known to have built-in split transactions. An advantage of our model is that thanks to the programming abstraction, the designer of a bus master model is relieved of taking care of the split transaction handling.

3.5. Arbitration Modelling

This subsection is concerned with the mechanisms employed for modelling arbitration; the discussion is largely independent of arbitration policy. In CA models, a time or clock triggered arbitration process is executed once per cycle. An efficient CX model can limit arbitration under the assumption of a time-invariant arbitration protocol because the grant decision does not
change unless the state of the waiting and active transactions changes. Rearbitration needs to be performed only in simulation cycles in which a new transaction arrives to the bus (in case of delta cycles modelling combinational arbitration) or in which the currently active transaction is finished or split (allowing a waiting transaction to be granted the bus).

Rearbitration can be modelled with one or a combination of the following methods: If the bus model exports an interface, the interface methods, executed with the masters’ processes, may perform arbitration without the need for a simulation process context switch. This comes at the cost of multiple rearbitration if multiple masters issue transactions in the same cycle, and it is not possible if communication is via channels (e.g. tlm_fifo). In this case, the bus model needs a process that is triggered by incoming transaction messages and performs arbitration actively (cf. M1, M2 in Figure 3). Since each channel has an event of its own, this requires the overhead of creating or-event-lists to activate the arbitration process in SystemC. The number of events can be reduced to one for all incoming transactions by implementing the bus model itself as channel with interface methods that trigger an internal re-arbitration event (cf. event in Figure 3). Split or finished transactions can trigger the re-arbitration event or an individual event.

4. Simulation Performance Results

4.1. Experimental Setup

The experimental setup used for performance evaluation of the bus models includes two masters of different priority and one slave. The high priority master issues transactions of increasing burst length that may be split by the slave, a RAM model. The parameters of the bus model are chosen to reflect the cycle timing of the AMBA AHB protocol [1], and priority based arbitration has been modelled. All models have been compiled with the same options and have been simulated on a computer with Pentium M 1.66 GHz.

With this setup, four different bus models have been simulated: A model CX1 at the PV+T abstraction and a cycle-accurate model CA as reference points, and two cycle-approximate versions, CX2 and CX3 using different choices of the identified alternatives. CX2 is a model using tlm_fifo as channels while CX3 implements the TLM interfaces itself, using a single arbitration event.

4.2. Comparison of Different Models

Figure 4 shows the simulation performance, measured in the number of 32 bit bus words whose transmission is simulated per second of CPU time, for the four models and for bursts of different size. No transactions have been split in this simulation. All models exhibit a performance that increases with the burst size due to less simulation overhead for arbitration and switch between transactions per transmitted bus word. We can see that the performance of the models CX2 and CX3 is consistently higher (by an average factor of about 5) than CA, and that CX1 (PV+T) exceeds CX2 and CX3 performance by an average factor of about 10. Only at very short burst length CX3 performance exceeds CX1; the reason is that CX1 lacks some of the optimizations that have been made in CX3.

At short burst length, model CX3 has a significant advantage over CX2, which diminishes towards larger bursts. The reason for this model behaviour is that the CX3 optimization of avoiding tlm_fifo and using just a single event is more significant when simulating short bursts requiring a higher rate of channel accesses and events.
4.3. Pre-emption Dependency

Different from PV+T, models CX2 and CX3 can simulate the pre-emption of transactions. To measure the effect of pre-emption on simulation performance, we have parameterized the slave model so that it randomly splits transactions. The percentage of bus word transfers which are split (i.e., multiple splits of a single transaction are possible) has been varied from 0% to 50%. Figure 5 shows the resulting simulation performance for model CX2. Performance degrades with increasing pre-emption ratio. It is reduced by a factor of up to 10 for long bursts and 50% pre-emption, compared to the non-preemptive case.

![Figure 5: CX2 performance with pre-emption](image)

The same measurement has been performed using model CX3, with results shown in Figure 6. Performance is generally higher compared to CX2, and the degradation factor due to pre-emption of bursts is down to a maximum of about 3. This is again due to the optimized implementation of model CX3, which also reduces the overhead of performing re arbitration in the case of transaction pre-emption and completion.

![Figure 6: CX3 performance with pre-emption](image)

5. Conclusions

We have shown the design of a cycle-approximate model that covers all bus features and represents transaction in an almost cycle-accurate way. The simulation of this model is by a factor of 5 faster than a cycle-accurate model and by a factor of 10 slower than a PV+T model that does not cover transaction pre-emption. We argue that modelling at an accuracy level between PV+T and CA is useful for architectural exploration because it permits significantly more precise estimation than PV+T. Therefore, a CX abstraction level should complement the other levels instead of being dropped, which appears to have happened in SystemC TLM standardization.

References