A Formal Full Bus TLM Modeling for Fast and Accurate Contention Analysis

Mao-Lin Li, Chen-Kang Lo, Li-Chun Chen, Ren-Song Tsay
The Department of Computer Science
National Tsing Hua University, Taiwan
Hsin-Chu City
Tel : +886-3-571-5131
e-mail : {mlli, cklo, lcchen,rstsay}@cs.nthu.edu.tw

Hong-Jie Huang, Jen-Chieh Yeh
Industrial Technology Research Institute, Taiwan
Hsin-Chu City
Tel : +886-3-582-0100
Fax : +886-3-582-0045
e-mail : { giffea, jcyeh}@itri.org.tw

Abstract*—This paper presents an effective Cycle-count Accurate Transaction level (CCA-TLM) full bus modeling and simulation technique. Using the two-phase arbiter and master-slave models, an FSM-based Composite Master-Slave-pair and Arbiter Transaction (CMSAT) model is proposed for efficient and accurate dynamic simulations. This approach is particularly effective for bus architecture validation and contention analysis of complex Multi-Processor System-on-Chip (MPSoC) designs. The experimental results show that the proposed approach performs 23 times faster than the Cycle-Accurate (CA) bus model while maintaining 100% accurate timing information at every transaction boundary.

I. Introduction

Due to the relentless demands for high-performance computation and low power consumption in embedded systems, multi-processor system-on-chip (MPSoC) has become the mainstream design approach. For MPSoC design, one of the most critical issues is the on-chip communication design (e.g., shared bus, bus matrix) because of the multiplied data exchange rate among the large number of components. As design complexity continues to increase, having an efficient and effective tool for early-stage system verification and validation is indispensible before committing a design to real hardware.

For communication architecture validation, designers are particularly interested in the rate of bus contentions and the effectiveness of contention handling. In practice, an arbiter is used to resolve contentions and determine transaction execution order according to certain arbitration policy, such as the round-robin (RR) or fixed priority (FP) policy. Contentions cause certain transactions to change or defer their execution order. Hence, accurate contention analysis is essential for performance evaluation during validation.

For early-stage validation, designers demand accurate contention analysis, correctness verification, and performance estimates by efficient system simulation. However, the complexity of traditional RTL simulation approaches makes these procedures prohibitively difficult. The transaction-level modeling (TLM) approach [1], which raises the abstraction level to speed up simulation performance, has been proposed as a solution.

To accurately simulate bus behaviors, traditional TLM bus modeling approaches adopt fine-grained models, such as cycle-accurate (CA) models, which simulate arbitration behaviors cycle by cycle. The heavy simulation overhead associated with these fine-grained approaches for handling the interactions between bus transactions and the arbiter limits the practicality of such approaches.

In contrast, for better performance, some researchers embrace coarse-grained modeling approaches, such as functional-level or cycle-approximate (CX) modeling [6, 11, 14]. However, these approaches can be misleading when used for validation purposes when arbitration information is inaccurate or missing.

Although various TLM bus models have been proposed, none can accurately perform arbitration analysis with efficiency. The main challenge is that the arbitration behaviors are irregular and unpredictable due to complicated combinations of requests and arbitration policy.

To effectively and accurately capture the timing behaviors of arbitration, we propose a *Two-Phase* arbiter model to abstract the procedure of arbitration and bus transactions in this paper. The arbitration is a dynamic handshaking process that can be split into *Request phase* and *Grant phase* according to the specific handshake signals controlling arbitration. Since the *Request phase* and *Grant phase* alternate repeatedly and synchronously with bus transactions, we can utilize the repetition property to pre-analyze the arbitration procedure without cycle-by-cycle simulation and guarantee the correct transaction execution order and the accurate handshaking process.

Furthermore, we extend the Finite State Machine (FSM)-based formal model proposed by Lo [4], which employs the regularity of bus transaction to statically analyze accurate cycle counts of each *Master-Slave-pair* bus transaction. Combining the above two abstracted models, we then have a formal Composite Master-Slave-pair and Arbiter Transaction (CMSAT) model to statically capture complete arbitrated bus transaction behaviors.

With the proposed CMSAT model, a dynamic simulation algorithm is designed to simulate the interleaving of *Request phase* and *Grant phase* effectively in order to maintain correct arbitration results without incurring redundant simulation overhead.

We implemented and tested the proposed approach on a few real designs. The encouraging experimental results demonstrate 23 times better performance than CA bus models while maintaining 100% accurate timing results in terms of cycle counts.

The rest of the paper is organized as follows. In section 2, we first review related work. Then, section 3 introduces a formal approach for describing the generic bus model. The proposed CMSAT model generation is then presented in section 4 and the experimental results in section 5. Finally, section 6 concludes this paper.

II. Related Work

The TLM idea has been deemed the most promising solution to system-level modeling problem. TLM provides system designers components and bus models of various abstraction levels. To implement the abstraction models, a system-level description language such as SystemC [6] is used. For example, Caldari et al. [7]

apply the concept of program state machine and implement it in SystemC to establish a cycle-accurate AMBA bus model [5].

Coware [9] proposes a cycle-accurate-level proprietary modeling library for several different bus architectures, such as Advanced Micro-controller Bus Architecture (AMBA) [5] and Open Core Protocol (OCP) [8]. Through a cycle-by-cycle simulation approach, it can precisely reflect the effect of arbitration behavior and offer high simulation accuracy. However, due to considerable simulation overhead, the performance may not be acceptable for architecture validation of complex MPSoC designs.

To improve simulation performance, Pasricha et al. [3] propose a cycle-count-accurate at transaction boundary (CCATB) approach, which eliminates unnecessary computations within a bus transaction while computing the precise cycle count. However, only 55% performance speedup over the pin-accurate bus cycle-accurate (PA-BCA) modeling approach is reported. To include arbitration and accurately resolve contentions, their approach essentially needs to check for bus requests at every cycle and hence no significant performance improvement is achieved.

To seek even better simulation performance, some researchers use simplified read/write functions and abstract arbitration procedures with approximate timing to represent various types of bus transfers (e.g., burst). The resultant cycle-approximate (CX) bus models have fewer simulation details and hence achieve higher simulation performance [6, 11, 14]. These approaches are suitable for fast early system prototyping. However, because of the inaccuracy, arbitration policy cannot be faithfully exercised and therefore the CX approaches are not reliable for communication architecture validation.

Although the above modeling techniques each meet certain design requirements, they lack of a systematic modeling mechanism. Therefore, D'sliva et al. propose synchronous protocol automata (SPA) [10] that systematically describe communication interfaces, including master, slave, arbiter and bridge, etc., for cycle-accurate level bus property verification.

Furthermore, Lo et al. [4] adopt the SPA modeling approach and automatically generate high-performance cycle-count-accurate composite master/slave models. However, the proposed method is good only for single master and single slave pair and it does not consider the effects of arbitration.

In contrast, we propose a two-phase (*Request phase* and *Grant phase)* arbitration model and extend Lo's model to cover multiple master-slave pairs. The proposed Composite Master-Slave-pair and Arbiter Transaction (CMSAT) model can effectively and efficiently capture full bus behaviors, including the most important contention effect. Furthermore, the cycle count of each phase can be obtained by analyzing FSMs, according to the repetition of the two-phase arbiter model, instead of cycle-by-cycle simulation. We simply manage the phase interleaving correctly and then can easily and accurately compute cycle count information of each phase, or full bus timing information, with no extra simulation overhead.

Before we introduce our proposed full bus simulation algorithm, we first present a generic bus modeling method.

III. GENERIC TLM BUS MODELING

A generic bus model involves multiple components (e.g., masters, slaves and arbiters). In MPSoC designs, it is common to have multiple bus requests contending for bus access at the same time. To resolve contention, an arbiter is implemented to perform arbitration. When arbitration is considered, the bus behavior becomes fairly complicated. In this section, we introduce a few definitions first and explain the basic idea of applying FSMs to our proposed approach. In the following, we will present a FSM-based formal model for communication interfaces description and then propose a simple two-phase arbiter model that can effectively and efficiently model a generic bus design.

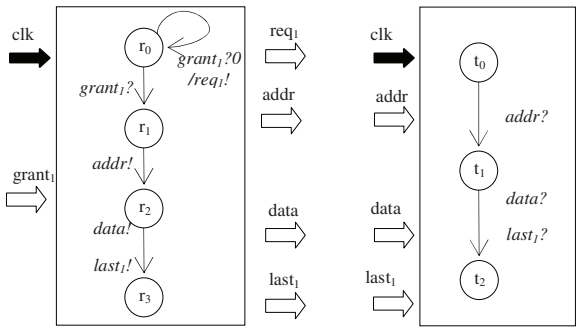

Figure 1: An example of *write* transaction described by FSMs.

A. A Formal Communication Interface Model

Before we formally specify the FSM-based communication interface model, we first illustrate in Fig. 1 a simple example to familiarize readers with basic FSM operations. The example shows a master and a slave interfaces described in FSMs performing a write transaction.

As shown in Fig. 1, the master and slave interfaces begin synchronously from state r_0 and state t_0, respectively. Initially, the master interface MI_1 is not granted, and it sends out the signal req_1 to request bus usage, denoted as $req_1!1$. Once MI_1 receives a grant signal to use bus, denoted as $grant_1?1$, it progresses its state from r_0 to r_1. Then, MI_1 emits *addr* (for data address, denoted as "*addr!*") to the engaged slave interface, and progresses the state transition from r_1 to r_2. Simultaneously, the engaged slave interface SI_1 receives the signal *addr*, denoted as "*addr?*", and then progresses its state from t_0 to t_1. This process continues until the state progress reaches the final states r_3 and t_2. At this point, the *write* transaction is completed.

Although communication interfaces are more than read/write operations, in practice *read* and *write* data transfers are the most basic communication behaviors. To describe a general and formal communication interface model, we modify the syntax of [10] and propose a definition in the following.

Definition 1: A Finite State Machine (FSM)-based communication interface model is a tuple (Q, Input, Output, C/O, V, $\rightarrow$, clk, q_0, q_f), where

1. Q: a finite set of states
2. Input: a set of input data and control signals
3. Output: a set of output data and control signals
4. C/O: condition/operation
5. V: a set of internal variables(e.g., the counter in burst transfer)
6. $\rightarrow \in$ Q x Q x C/O x clk?: transition relations
7. $q_0, q_f \in$ Q: the initial state and the final state

According to the above definition, the FSM for each communication interface has certain specified input and output signals and performs transitions between states listed in a set *Q*. The state transition in each FSM starts from the initial state q_0 and ends at the final state q_f. Every *clk* tick triggers a state progress. The operation *O* is a set of signal operations. For example, the action *"s!"* denotes that the signal *s* is emitted from the interface, and *"s?"* denotes that the signal *s* is read by the interface. C/O on each state progress edge indicates that once the condition *C* is met, the corresponding operation *O* will be issued. The condition *C* is checked against with the value of

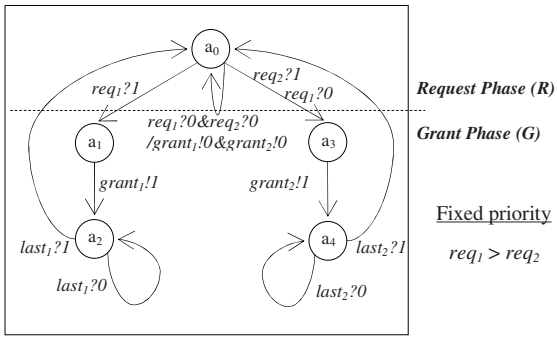

Figure 2: An example of a fixed priority arbiter described in FSM.

the internal variables in *V* (e.g., the counter in burst transfer) or specific input signals (e.g., *last*).

The above formal communication interface model describes only how one component communicates with others. In the next section, we extend the idea and explain how to model a generic bus.

B. A Two-Phase Arbiter Model

Like the formal communication interface model, the arbitration process can be described as an FSM. In general, the arbiter receives bus requests from master components and then arbitrates and grants bus access to one of the requests according to a designer-specified arbitration policy. The above arbitration procedure is accomplished by asserting specific handshake signals. Hence, we further divide the arbitration procedure into two phases, *Request phase (R)* and *Grant phase (G),* according to handshake signals that control arbitration. Before arbiter emits a grant signal, the states of arbiter are included in the *Request* phase. Once arbiter emits a grant signal, the arbiter's phase transits to the *Grant phase* and returns to the *Request phase* after the granted transaction finishes its data transfer. Essentially, at the *Request phase* the arbiter receives external requests and selects a master-slave pair for bus transaction while at the *Grant phase* the granted master-slave pair executes the transaction.

The example in Fig. 2 illustrates an arbiter FSM which adopts a *fixed priority* arbitration policy. We assume that the request req_1 from MI_1 has higher priority than req_2 and the fact is reflected in the arbiter FSM.

We first explain how the request and grant procedures work. In example Fig. 2, the state of arbiter will be a_0 initially. The annotation *"req_1?1"* on the state transition edge from a_0 to a_1 indicates that the arbiter receives a bus request from MI_1. Similarly, *"req_2?1, req_1?0"* on the transition from a_0 to a_3 indicates that the request from MI_2 is asserted while MI_1 has no request. In general, in the *Request phase*, the arbiter collects all incoming request signals and computes which master is granted.

After the *Request phase*, a master is selected and then the arbiter moves to *Grant phase* and assigns the master to have the bus for data transfer. In Fig. 2, when req_1 is asserted, according to the arbitration policy the request from MI_1 has the priority and hence the arbiter asserts $grant_1$, or *"$grant_1$!1"*, and grants MI_1 to start its data transfer.

After MI_1 finishes its transaction, it sends a notification signal, $last_1$ to the arbiter, denoted as *"$last_1$?1",* and has the arbiter return to its initial state a_0 and get ready for next request processing.

If only req_2 from MI_2 is asserted and req_1 is absent, the arbiter will grant MI_2 for data transfer and the granting process is similar to what have been described for MI_1. Furthermore, if no request tends to use the bus, the arbiter stays in the initial state a_0.

After the *Grant phase* is completed, the arbiter returns to the *Request phase*. The two phases alternate repeatedly throughout the system active period for bus transactions. In fact, an arbiter functions exactly as a scheduler. It collects issued requests and grants one for execution according to the arbitration policy designed in terms of the arbiter FSM.

One key point is that with the proposed two-phase arbiter model, the state progression of an arbiter can be greatly simplified without losing functionality or timing correctness. We will elaborate on this after we define a formal model for generic buses.

After adding the arbiter model along with the master and slave models, we now can define a formal generic bus model. And in the following section, we propose a static model abstraction and dynamic simulation algorithm leveraging the two-phase arbiter model to achieve fast and accurate full bus simulation.

IV. CMSAT Model Generation

In this section, we further elaborate our main idea and demonstrate the effectiveness of our approach. The approach has two steps: static model abstraction and dynamic simulation. At the static phase, we analyze the behaviors of bus transactions and arbitration process and create abstract models by optimizing routine simulation procedures. Then at the dynamic simulation phase, with the interacting signals and actual data we compute accurate arbitration and bus transaction results.

A. Static model abstraction

We now explain the concept of static model abstraction. First, we review the master-slave pair model, and then the arbiter model and finally the composite model.

1) Master-slave pair model compression

The basic bus function is essentially data transfer, or data read/write, between masters and slaves. In this paper, we adopt and extend Lo's compression approach for model abstraction of the master/slave transaction pair with accurate cycle count information retained [4].

Basically, the compression algorithm analyzes the FSM-pair of master/slave interfaces and merging them into one FSM that represents the behavior of bus transaction. The compressed FSM *eliminates* confirmed internal handshaking signals between master and slave interfaces and reduces unnecessary simulation overhead with fewer transition steps while maintaining same cycle count information as the CA model.

On the other hand, we preserve the external interacting signals, such as the handshaking signals *req*, *grant* and *last,* which interact with the arbiter for accurate dynamic behavior simulation.

The FSM shown in Fig. 3(a) is the compressed write transaction model of the master-slave pair discussed in Fig. 1. The address and data transfers are compressed into one state transition step with a computed cycle count equivalent to the actual number of cycles taken. Note that each rhombus in the compressed model denotes a composite FSM node. Details about the compression algorithm can be found in [4].

With the compressed bus model, once the issued bus transaction is granted during simulation, the cycle count of each bus transaction is readily obtained without the need to do slow cycle-by-cycle simulation. Simulation performance, hence, is significantly improved.

The compressed bus transaction model is defined as follows.

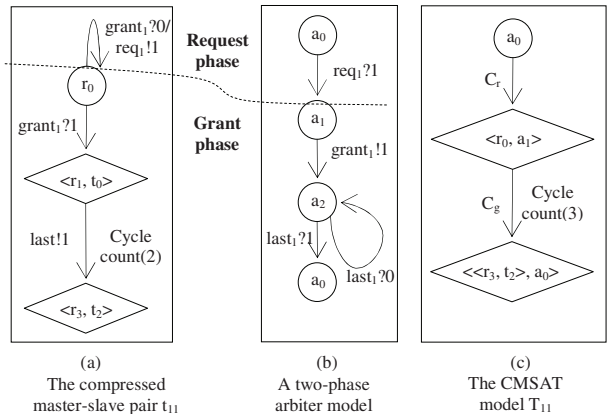

Figure 3: An example of static model abstraction for bus transaction and arbiter.

Definition 2: A *compressed bus transaction* model t_{ij} is a merged FSM of a master-slave interface pair generated from the compression algorithm, or $t_{ij} = (m_i \,||\, s_j)$, where

t_{ij}: the compressed bus transaction model of the pair of m_i and s_j.
m_i : the i-th master interface in the bus;
s_j: the j-th slave interface in the bus;
||: compression function;

2) The composite master-slave pair and arbiter transaction (CMSAT) model

In fact, bus transactions and arbitration process are both FSMs synchronized by specific handshaking signals. Moreover, each master-slave pair bus transaction can also be divided into two phases, *Request phase* and *Grant phase*, and matches the two-phase arbiter model perfectly.

As illustrated in Fig. 3(a), if the compressed master-slave bus transaction model t_{11} is activated, it will continue asserting the request signal ($req_1!1$) until it receives a grant ($grant_1?1$). This portion is clearly in the *Request phase*. After being granted, it enters the *Grant phase*. It then starts data transfer and after completion it sends out a finish notification ($last_1$) before returning to the request phase.

To focus on the arbitration process analysis for req_1, we show in Fig. 3(b) a partial FSM of the arbiter from Fig. 2 related to req_1, $grant_1$ and $last_1$. Once the arbiter is in the *Request phase,* it checks if any request signal is asserted. Following assumed priority policy, when the arbiter detects that req_1 is asserted, it takes one cycle arbitration time and asserts a corresponding grant signal ($grant_1!1$). It then waits for the finish notification ($last_1$) from t_{11} before it returns to the *Request phase*.

Normally the arbiter *Request phase* takes a fixed computation time to handle received requests. The request processing time in general can be pre-analyzed based on the combination of requests. If not, we simply compute the arbitration time in terms of cycle count (C_r) at runtime. For the fixed-priority case in Fig. 2, the request always takes arbiter one cycle time to process grant.

While in the *Grant phase*, the arbiter simply waits for the granted bus transaction finishing data transfer before entering next request phase. In fact, the granted master-slave pair and the arbiter are progressing synchronously and hence we can further composite the master-slave pair and the arbiter model into an optimized CMSAT model for full bus simulation. After composition, the internal handshaking signals, such as grant signal and bus transaction completion signal, between the active master-slave pair and the arbiter can be *eliminated* following Lo's compression algorithm. At the same time, the cycle count of grant phase (C_g) is statically calculated.

The resultant CMSAT model shown in Fig. 3(c) is the composition of the master-slave pair in Fig. 3(a) and the two-phase arbiter model in Fig. 3(b). Note that in the CMSAT model the handshaking signals, $grant_1$ and $last_1$, are eliminated and the grant phase is determined to consume three cycles, comprising one cycle for the arbiter asserting $grant_1$ and two cycles for bus data transfer.

We now formally define the composite master-slave and arbiter transaction (CMSAT) model in the following.

Definition 3: The composition of a compressed bus transaction t_{ij} and a two-phase arbiter model A is denoted as $T_{ij} = (t_{ij} \,||\, A)$, where

T_{ij} : the composite model of t_{ij} and A.
t_{ij} : the compressed bus transaction of the pair of m_i and s_j;
A: the two-phase arbiter model described in FSM;
||: compression function;

Each CMSAT model represents a complete process for the arbiter granting a specific request and returning to next request phase after the granted bus transaction is finished. This optimized model eliminates unnecessary simulation overhead and hence leads to high performance simulation.

Next, we discuss how to apply CMSAT models at the dynamic simulation phase.

B. Full-bus dynamic simulation

The key for the cycle-count-accurate full bus simulation to correctly simulate contention behaviors is to maintain a correct bus transaction execution order. Then, with the CMSAT model, accurate transaction execution cycle counts are efficiently computed.

In practice, virtually all bus requests can be viewed as being stored in a request queue waiting for arbitration. After a request is granted for bus transaction, the remainders stay in the queue and the granted request will start bus transaction until completion. Furthermore, at the completion of the granted request, only the requesting master or the accessed slave (if it is also a master) may generate later new requests and affect arbitration subsequently. Hence, we can check the master and the slave of the granted request at the completion time point and determine whether any new requests should be added into the queue.

To make the simulation process efficient, in implementation we extend the request queue to include also *future* requests. Nevertheless, the arbitration procedure processes only the *active* requests which are initiated before the arbitration starting time.

We now illustrate our algorithm using an example in Fig. 4 with the fixed-priority arbiter in Fig. 2. At first, assume that both req_1 and req_2 are simultaneously active at t_1 and are inserted into the request queue. The arbiter first advances to time t_1, the earliest time new

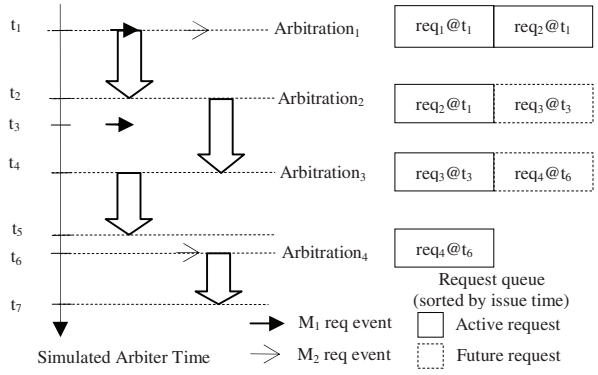

Figure 4: An example of dynamic simulation.

requests occur. Then the arbiter grants req_1 according to the specified arbiter model ($Arbitration_1$). Consequently, the corresponding CMSAT model of req_1 is selected and then its C_r and C_g are computed accordingly. In contrast, req_2 is still stored in the request queue since it is not granted and cannot be executed.

We check if M_1 or S_1 will generate new requests at t_2, the completion time of req_1, which is activated from master M_1 to slave S_1. Suppose that a new request req_3 is generated at time t_3. Then this *future* request is inserted into the request queue. Now by advancing the arbiter time to t_2, the completion time of req_1, another run of arbitration process begins ($Arbitration_2$). At this moment, the arbiter finds that only req_2 is active in the queue and hence grants req_2 for execution.

Assume that req_2 finishes its transaction at time t_4, and then we check if M_2 has a new request generated and find that it does generate a new request req_4 at time t_6, which is inserted into the request queue as a *future* request.

Now at time t_4, the arbiter starts another arbitration process ($Arbitration_3$) and finds that req_3 at t_3 is the only *active* request and hence grants req_3 for execution.

Assume that at time t_5, req_3 finishes execution and M_1 does not generate a new request. Then, when the arbiter tries to start a new run of arbitration processes, it finds that there is no *active* request but only one *future* request req_4 at t_6. Therefore, the arbiter sets the new arbitration time to t_6 and determines to grant req_4, which completes its transaction at time t_7.

The above illustrative cases cover most arbitration situations. A more general and formal full bus simulation algorithm is proposed in the following.

Procedure *Full_Bus_Simulation()*

0. **Init:** Generate the CMSAT models of the arbiter and all master-slave pairs.
1. **S**et the arbiter time to 0 and the request queue to empty. For each master, we compute the first request and insert the request into the request queue.
2. **Do until** the request queue is empty.
3. **If** no *active* request in the request queue
   a. Advance the arbiter time to the request time of the earliest *future* request.
4. **Else**
   a. Select and grant an *active* request following the given arbitration policy.
   b. Compute the *Request phase* execution time C_r of the active request.
   c. Compute the *Grant phase* execution time C_g according to the CMSAT model of the active request.
   d. Update the arbiter time by adding C_r and C_g to the current arbiter time.
   e. Examine the requesting master and accessed slave of the granted request, if any of them will generate new request, push the request into the request queue.

We use the request queue to preserve the requesting order and apply the CMSAT models to calculate accurately timing information rapidly until the request queue is empty. Our approach achieves an effective full-bus simulation without need to do cycle-by-cycle simulation. Moreover, the algorithm can be implemented in POSIX pthread or common simulation engine, e.g., SystemC. Each transaction is represented as an individual process and can look ahead to determine whether new requests will be generated at the end of the transaction.

C. Discussions

The main assumption of the proposed CMSAT model is that once a transaction enters into the *Grant phase*, it cannot be preempted and no other transactions on the same bus can enter the grant phase until it returns to the *Request phase* again.

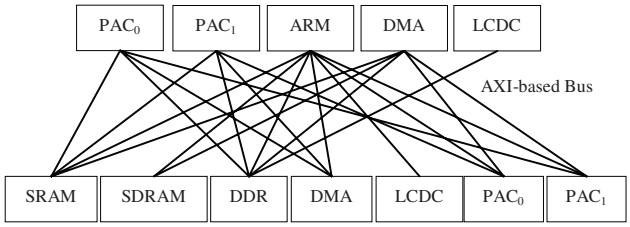

Figure 5: The modeled PAC-Duo platform.

To the best of our knowledge, due to the complexity and limited benefits of the circuit design, most practical designs do not allow preempting request to terminate a transaction in the middle of data transfer (e.g., address, data…). This observation is confirmed with a bus protocol survey in [13].

In practice, bus preemption can still occur at the end of transaction execution. Masters such as DMA (Direct Memory Access) may request multiple transactions at a time. For this type of requests, the preempted master is designed to complete its current transaction before handing over the bus to the preempting master. This preemption case can be handled perfectly with our proposed algorithm, since the arbitration is performed at the phase boundaries.

V. EXPERIMENTS

To demonstrate the effectiveness of our methodology, we apply our modeling and simulation approach on the AMBA AXI-based bus matrix of the Parallel Architecture Core Duo (PAC-Duo) platform from ITRI [12]. We compare the simulation performance and accuracy of our model with the CA model provided by Coware [9], a popular commercial tool.

A. Modeling bus matrix of PAC-Duo platform

The diagram in Fig. 5 shows the PAC-Duo platform according to our formal definition. It consists of two PAC DSP processors, an ARM processor, a DMA, LCDC (LCD controller), and memories. The AXI-based bus matrix of the platform is modeled through the proposed approach, while all IP components are cycle-count accurate TLM models [15].

To test the practicability of our bus modeling approach, we run an H.264 decoder application with a QVGA video stream (320x240 per frame) on the platform. The application flow starts by having the ARM processor load H.264 decoder program from SRAM and configure the PAC-DSP processors for H.264 decoder execution. The two DSP processors decode the H.264 frames in a pipeline fashion, while DMA helps with image data transfers. Whenever a frame is finished decoding, the ARM processor configures LCDC to read and display the frame.

B. Accuracy and performance comparison

To confirm the accuracy of our approach, we verify that all the transaction execution beginning and end time points of our bus model are the same as that of the CoWare CA model.

For simulation performance evaluation, Table 1 lists the performance comparison in terms of the number of transactions per second. The whole platform simulation of our bus model is 5.2 times faster than the platform of CoWare CA AXI bus model with the same IPs. For clear comparison, our bus model is 23 times faster than the Coware CA model if evaluating only on bus execution time.

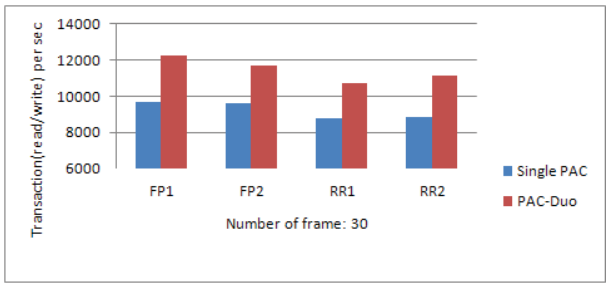

Figure 6: Architecture performance evaluation with different arbitration arbitration policy

This huge performance improvement is mainly gained from the static analysis of CMSAT model generation. Particularly, for burst-based bus protocols, such as AXI, simulation performance is significantly improved since most simulation overhead from the cycle-by-cycle data transfer and handshaking with the arbiter are eliminated by static analysis.

TABLE 1: The performance (transactions per sec) comparison

	Whole platform performance / Speedup	**Communication performance / Speedup**
Coware CA	598/ 1X	708/ 1X
CMSAT	3121/ 5.2X	16500/ 23X

C. *Architecture performance evaluation*

Finally, we demonstrate bus architecture performance evaluation for the PAC-Duo platform. We evaluate the effect of arbitration policy by examining four different arbitration policies–a fixed priority policy where DMA is of higher priority than LCDC (FP_1), another fixed priority policy where LCDC is of higher priority than DMA (FP_2), a Round Robin policy with 25 cycles time slot (RR_1) and another Round Robin policy with 30 cycles time slot (RR_2).

Fig. 6 shows the results of total throughputs of the platform with the above four different arbitration policies. In addition, a modified platform with only one PAC DSP is listed for reference. It is found that the PAC-Duo platform outperforms the single PAC platform, but the Duo platform is more sensitive to the choice of arbitration policy. For the PAC-Duo platform, performance can differ as much as 15% depending on the choice of arbitration policy, while for the single PAC platform the difference is only 9%. This is due to the fact that the PAC-Duo platform has a much higher contention rate because there are more active masters requesting data transfers.

Through the experiments, we have demonstrated that our proposed approach can efficiently and effectively optimize bus architecture design. Our approach requires very little modeling effort. It needs only a few more lines of system description while the corresponding bus models and arbiter models are automatically generated.

VI. CONCLUSION

In this paper, we have presented a highly efficient FSM-based Composite Master-Slave pair and Arbiter Transaction (CMSAT) model for full bus simulation. Following the proposed approach, designers can easily describe bus designs and perform Cycle-count Accurate (CCA) simulation for full bus performance evaluation and architecture validation.

Our approach can handle most bus architectures except some advanced features such as out-of-order transfer, which we will address in future work.

VII. REFERENCES

[1] L. Cai, D. Gaski. "Transaction Level Modeling: An Overview," in CODES+ISSS, 2003

[2] T. Grotker, S.Laio, G. Martin, S. Swan, *System Design with SystemC*, Kluwer Academic Publishers, 2002.

[3] S. Pasricha, N. Dutt, M. Ben-Romdhane, "Extending the Transaction Level Modeling Approach for Fast Communication Architecture Exploration," in DAC, 2004

[4] C. K. Lo, R. S. Tsay, "Automatic Generation of Cycle Accurate and Cycle Count Accurate Transaction Level Bus Models from a Formal Model," in ASP-DAC, 2009

[5] ARM Ltd. AMBA Protocol Specification. www.arm.com

[6] Open SystemC Initiative (OSCI). SystemC 2.2.0 Documentation. www.systemc.org

[7] M. Caldari, et al., "Transaction-Level Models for AMBA Bus Architecture Using SystemC 2.0" in DATE, 2003

[8] Open Core Protocol International Partnership (OCP-IP). www.ocpip.org.

[9] Coware. www.synopsys.com

[10] V. D'silva, S. Ramesh, and A. Sowmya, "Synchronous Protocol Automata: A Framework for Modeling and Verification of SoC Communication Architecture", in DATE, 2004

[11] M. Radetzki, R. Salimi Khaligh, "Modelling Alternatives for Cycle Approximate Bus TLMs," in Proc. Forum on Design Languages(FDL), 2007

[12] Z. M. Hsu, J. C. Yeh, I. Y. Chuang, "An Accurate System Architecture Refinement Methodology with Mixed Abstraction-Level Virtual Platform", in DATE, 2010

[13] W. Klingauf, R. Gunzel, O. Bringmann, P. Parfuntascus, and M. Burton, "GreenBus: a generic interconnect fabric for transaction level modeling ", in DAC , 2006

[14] R. B. Atitallah, S. Niar, S. Meftali, and J. L. Dekyser, "An MPSoC Performance Estimation Framework Using Transaction Level Modeling", in RTCSA, 2007

[15] C. K. Lo, L. C. Chen, M. H. Wu, R. S. Tsay, "Cycle-count-accurate Processor Modeling for Fast and Accurate System-level Simulation," in DATE, 2011