Cross-abstraction Functional Verification and Performance Analysis of Chip Multiprocessor Designs

Gabor Madl, *Student Member, IEEE,* Sudeep Pasricha, *Member, IEEE,*
Nikil Dutt, *Fellow, IEEE,* and Sherif Abdelwahed, *Senior Member, IEEE*

***Abstract*—This paper introduces the *Cross-abstraction Real-time Analysis* (CARTA) framework for the model-based functional verification and performance estimation of chip multiprocessors (CMP) utilizing bus matrix (cross-bar switch) interconnection networks. We argue that the inherent complexity in CMP designs requires the synergistic use of various models of computation to efficiently manage the trade-offs between accuracy and complexity. Our approach builds on domain-specific modeling languages (DSMLs) driving an open-source tool-chain that provides a cross-abstraction bridge between the finite state machine (FSM), discrete event (DE) and timed automata (TA) models of computation, and utilizes multiple model checkers to analyze formal properties at the cycle-accurate and transaction-level abstractions. The cross-abstraction analysis exploits accuracy for functional verification, and achieves significant speedups for performance estimation with marginal accuracy loss. We demonstrate results on an industrial strength networking CMP design utilizing a bus matrix interconnection network. To the best of our knowledge, the CARTA framework is the first model-based tool-chain that utilizes multiple abstractions and model checkers for the comprehensive and formal functional verification, performance estimation, and real-time verification of bus matrix based CMP designs.**

***Index Terms*—Real-time, performance analysis, model checking, chip multiprocessor, bus matrix interconnect.**

I. INTRODUCTION

Modern *chip multiprocessors (CMPs)* consist of several heterogeneous components such as programmable processors, custom logic blocks, memories, and peripherals, all of which are connected together via an interconnection network. These CMP designs must satisfy increasingly complex performance constraints for emerging applications. This is becoming more and more challenging for system designers because of the large number of components on a chip that have multifaceted dependencies and interactions with each other. Model-based design is an emerging paradigm that aims to manage this complexity by systematically capturing key properties of CMPs, such as their structure, parameters of individual components, and their interactions.

This paper introduces the model-based *Cross-abstraction Real-time Analysis (*CARTA*)* framework for the cross-abstraction analysis of CMP designs, that combines the concepts of component-based design, domain-specific modeling, simulations, and model checking to provide a unified framework for the functional and performance analysis of CMPs with bus matrix interconnection networks. The design flow aims to address three major challenges in the formal analysis of CMP designs: *(1) functional verification* - to ensure that the system will not be trapped in a deadlock or livelock state, *(2) performance estimation* - in order to obtain tight bounds on the worst case performance of the CMP design, and *(3) verification of real-time properties* - to prove whether individual deadlines for tasks and performance estimates hold for the CMP design.

The proposed approach builds on several methods originally developed for the real-time verification and performance estimation of software-intensive *distributed real-time embedded (DRE)* systems. While CMP designs themselves can be viewed as DRE systems, the communication subsystem in CMP designs has a major impact on both design and analysis. Unlike software-intensive DRE systems that communicate over packet-switched networks, CMP designs often utilize complex bus matrix architectures, where access to the bus is managed by an *arbiter* (or several arbiters). Bus protocols and arbitration policies have a major impact on key de-

This work was partially supported by the NSF grants CCR-0225610, ACI-0204028, CNS-0615438, CNS-0613971, NSF SOD grant CNS-0804230, a CPCC Fellowship, and a grant by Fujitsu Laboratories of America.

sign parameters such as throughput and delays, and present new challenges for functional verification. In particular, deadlock-freedom and livelock-freedom is not guaranteed by bus protocols, but is a key requirement for designers. A key contribution of this paper is to show how methods for the analysis of DRE systems can be adapted to CMP designs utilizing fully connected bus matrix interconnects, and how point arbitration policies can be expressed by the non-preemptive scheduling of task graphs. The contributions of this paper are focused on the following areas:

- We describe the CARTA framework for the cross-abstraction analysis of CMP designs, and define a model-based design flow for the analysis of CMP designs in Section II.
- We utilize the ALDERIS *domain-specific modeling language (DSML)* introduced in [1] for the formal modeling of CMPs with bus matrix interconnection networks. ALDERIS was proposed as a DSML for the modeling of software-intensive DRE systems. In this paper we extend the use of ALDERIS to complex bus matrix architectures commonly used in modern CMPs. The novelty in this paper is to show how complex bus designs can be abstracted out as transaction-level models, and how we translate resource allocation to the ALDERIS task graph model. The ALDERIS DSML is used as a high-level specification of the CMP, and directly drives the functional verification, performance estimation, and performance verification methods. This is described in more detail in Section IV.
- We describe an approach for the functional verification of CMPs using the *ARM Advanced Microcontroller Bus Architecture Advanced High-speed Bus (AMBA AHB)* bus matrix interconnection networks (also referred to as Multi-layer AHB and AHB-Lite). We extend earlier work on the verification of simple bus designs [2] to complex bus matrix structures. We use FSM models of the AMBA AHB protocol with cycle-accurate timing information to formally verify deadlock-freedom. We describe this method in Section V.
- We utilize a *discrete event simulation (DES)*-based formal performance estimation method described in [3] to estimate the real-time performance of a CMP design. By switching to more abstract representation of the design, we achieve significant speedups and scalability increase with negligible accuracy loss. Section VI describes results for this work.
- We build on our earlier work on the real-time verification of *distributed real-time embedded (DRE)* systems [4], [5], [6], [7] to propose a method to verify estimates for real-time performance using timed automata model checkers. By incorporating timed automata model checkers in the design flow, we can prove that the model satisfies the performance estimates achieved by the DES-based method. Section VII presents this timed automata-based real-time verification method.
- Finally, the major contribution of this paper is that it tightly integrates all of the above steps in the CARTA model-driven design analysis framework. This approach provides a way to use time-accurate models for functional verification, and more abstract representations for scalable performance estimation. By adopting the right abstractions to different steps of the analysis, we can significantly increase scalability when needed, while also retaining accuracy for steps where it is needed. We compare the scalability of the proposed methods in Section VIII.

II. THE CARTA FRAMEWORK

Fig. 2 shows our proposed cross-abstraction real-time analysis framework that provides a way to utilize the right level of abstraction for each analysis method. The three challenges addressed by our proposed analysis framework – functional verification, performance estimation, real-time verification – require different approaches, models of computation, abstractions, and tools for formal analysis. We pick the model of computation and abstraction level for each analysis method that provides the most efficient analysis. Finding the right abstraction is a key challenge for the model-based analysis of CMP designs.

For *functional verification*, it is important to accurately capture the signals in the bus matrix interconnection network. If the analysis model is too abstract, certain problems can remain undetected in the design phase. In our framework, we make use

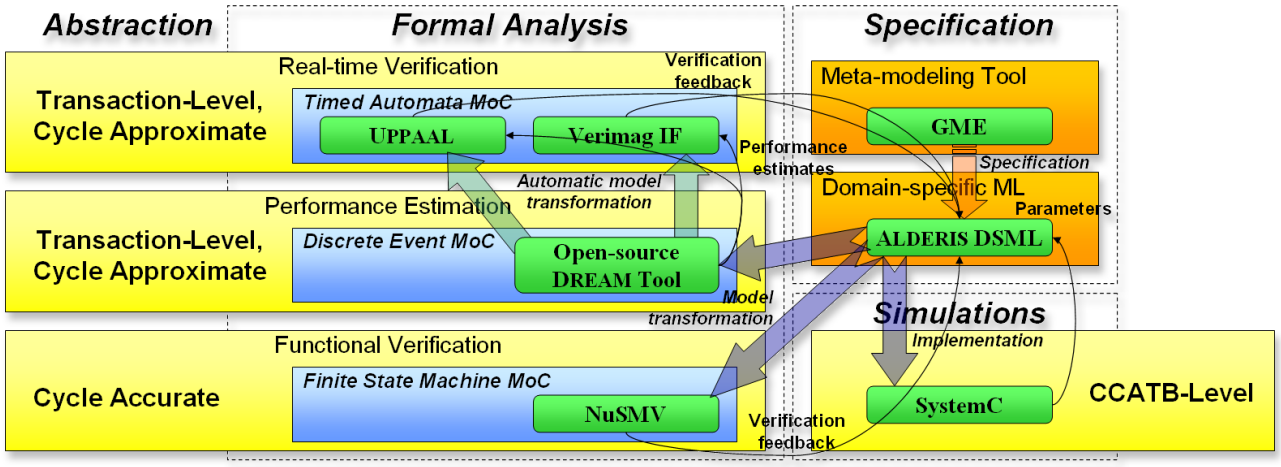

Fig. 1. The CARTA Model-based Analysis Framework

of a cycle-accurate *finite state machine (FSM) model of computation (MoC)* to capture the bus protocol, and arbitration algorithms. Using this model, we can efficiently check for all combinations of communication signals that satisfy the protocol to verify that no deadlocks and starvations occur in the CMP design.

The effectiveness of *performance estimation* and *real-time verification*, on the other hand, is primarily limited by scalability issues. Cycle-accurate FSM models quickly lead to the state space explosion problem, when used for performance estimation, or real-time verification. Therefore, we need to raise the abstraction to transaction-level formal models. Transaction-level abstractions are well-established in the domain of simulation-based design exploration [8]. Transaction-level formal models in our context are event-driven, and communicate via asynchronous message passing. Timing information in the models is captured as time intervals associated with events. We apply the transaction-level modeling concept to increase the scalability of formal methods in the CARTA framework.

The CARTA framework builds on various modeling and analysis tools created by the research community and the authors. We utilize the *Generic Modeling Environment (GME)* [9] as a modeling tool for designing ALDERIS [1] models (http://alderis.ics.uci.edu), as described in Section IV. *Domain-specific modeling languages (DSMLs)* in our approach are defined by the concept of model-integrated computing [10], that promotes the use of *meta-modeling* to create custom modeling languages which are a good fit for a specific problem domain. ALDERIS captures key properties of a CMP design, such as computation units, inter-component dependencies, the mapping of tasks to HW or SW, execution times and delays, and key constraints that the design has to satisfy. CMP designs are specified using the ALDERIS DSML, and drive the CARTA model-based analysis framework.

ALDERIS models of CMP designs are executable *discrete event (DE)* models with formal semantics. These models can be transformed into *finite state machine (FSM)* models with cycle-accurate timing accuracy. We utilize the open-source *Distributed Real-time Embedded Analysis Method* (DREAM) tool (http://dre.sourceforge.net) for the performance estimation of ALDERIS models, and the NuSMV [11] model checker for the functional analysis of the FSM models. The DREAM tool also generates a direct *timed automata (TA)* representation of ALDERIS models, that can be analyzed by the UPPAAL [12] and Verimag IF [13] timed automata model checkers.

A. Model-based Design Flow using CARTA

Fig. 2 shows the proposed model-based design flow using CARTA. The proposed approach is a multi-step process, in which the domain-specific model continually evolves until all required properties are satisfied. This evolution is performed by the designer based on feedback from the analysis tools. The design flow starts with the domain-specific model, capturing key properties of the design, such

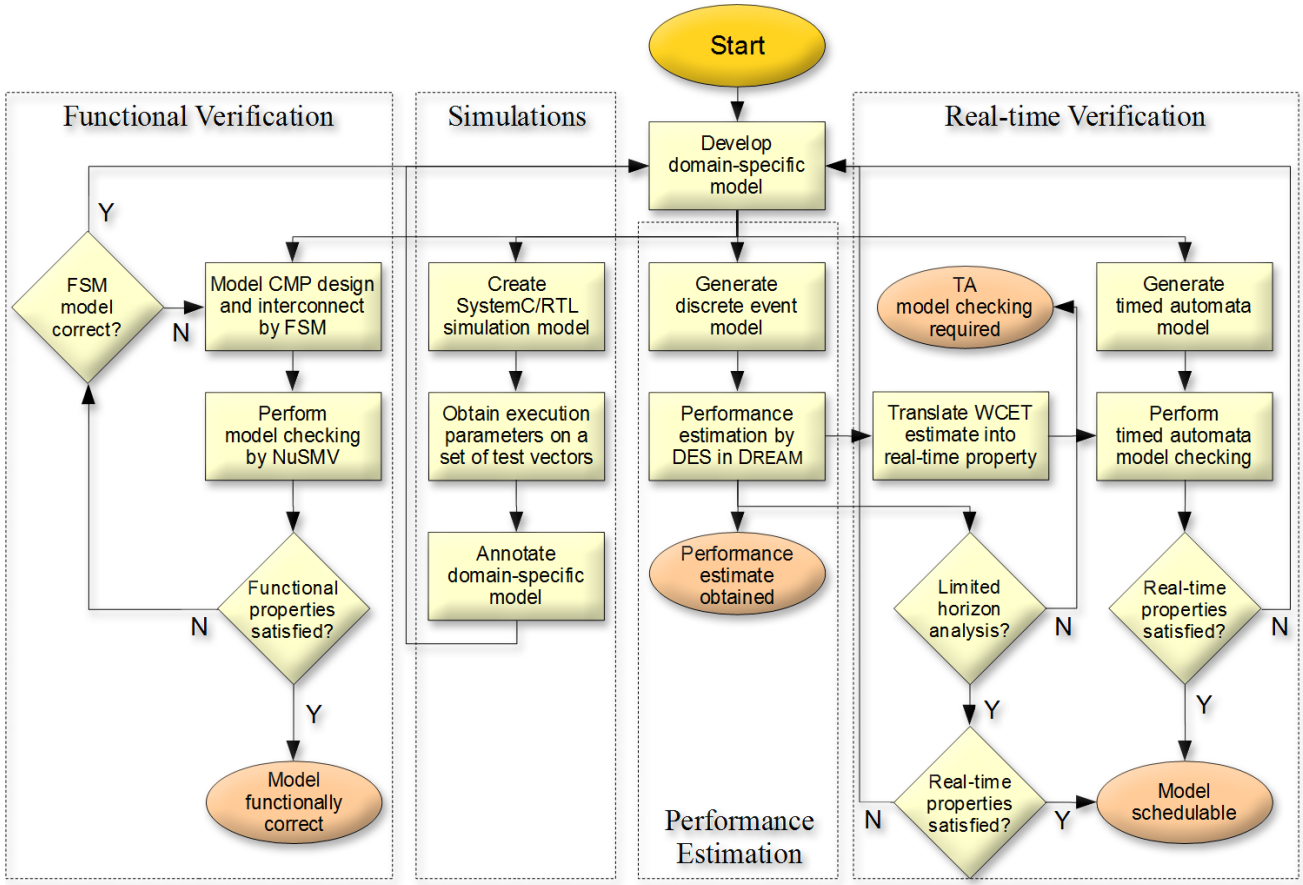

Fig. 2. Model-based Design-flow using CARTA

as its structure, behavior and environment. CARTA utilizes the ALDERIS DSML to specify CMP designs for real-time analysis. Analysis methods utilized by CARTA include *(1)* functional verification, *(2)* simulations, *(3)* performance estimation, and *(4)* real-time verification. While functional correctness and simulation results are required for performance estimation and real-time verification, some of the steps can be performed concurrently. This approach reduces the overall design time compared to a sequential design process.

The goal of *functional verification* is to prove that the CMP design is safe and works according to specification. CARTA proposes the use of cycle-accurate FSM models for functional verification by the NuSMV tool, and models masters and slaves connected to the interconnect as "black boxes", that communicate non-deterministically. The model checker explores the resulting state space to prove required properties in the design. Interconnect protocols have to be captured by a formal specification for model checking, therefore designers need to create models manually whenever a new protocol is considered for use in CMP designs. When working with a new protocol, if a property is not satisfied, designers need to check whether the problem is in the design itself, or in the manually created FSM models, and update models accordingly. A compositional approach for modeling is desirable, as it allows to reuse the formal specification for various CMP designs. In this paper we describe cycle-accurate FSM models for CMP components built on the AMBA AHB [14] protocol, and use these models for compositional functional verification.

Simulations form an integral part of the proposed design flow, and provide accurate task execution times and delays to the ALDERIS models. For the purposes of simulation, we capture CMP designs in the SystemC [15] modeling language at the *Cycle Count Accurate At Transaction Boundaries (CCATB)* modeling abstraction [16]. SystemC is a C++ library that provides a rich set of primitives for modeling communication and synchronization. The CCATB modeling abstraction is a form of

transaction-based bus cycle accurate model that enables fast and accurate performance estimation for CMP designs. CCATB captures transactions in the design using function calls which allow a significant speedup in simulation. For instance, in the AMBA AHB on-chip communication architecture, hundreds of signals can transition during a data read or write issued from a processor to a memory. In the CCATB model, a `read()` or `write()` function call captures the functionality of the hundreds of signals, while still maintaining cycle accuracy required for meaningful exploration. This leads to a reduction in modeling time, and improves simulation speed by several orders of magnitude over signal-accurate C++ or RTL models. CCATB also performs additional optimizations, such as effectively clustering static CMP delays and incrementing simulation time in chunks, to further improve simulation speed.

The CARTA framework facilitates a novel *discrete event simulation (DES)*-based simulation-guided *performance estimation* method introduced in [3]. Models in the DRE MoC capture dependencies between timers, tasks, channels in a formal setting, as well as timing information and the scheduling algorithm. Therefore, the dependencies essentially impose a partial ordering between events in the model. It was shown in [3] that two execution traces of the DRE MoC, that were obtained by DES are equivalent, if the total (untimed) ordering of events is the same. If the order of events is fixed, the DE model is deterministic, and one simulation is sufficient to verify real-time properties. When the order of events can change due to non-deterministic execution times, all the different traces need to be enumerated in order to verify that no deadlines are missed in the model. As long as the order of events is preserved, one only has to consider the largest possible timestamps of events. Thus, the DES-based performance estimation method is based on repeated simulations, that explore the non-deterministic ordering of events in the CMP models. We refer to this approach as "simulation-guided model checking".

The simulation-guided model checking method is built on the assumption that discrete event simulations using limited horizon are sufficient, after which the system returns to an initial idle state. This is often the case if computation is driven by periodic timers. If this assumption can be guaranteed, then the analysis is exhaustive, and proves the schedulability of the CMP design. If not, or the analysis does not terminate due to scalability issues, then an additional step is needed for *real-time verification* by timed automata.

We use results of the performance estimation to formalize properties on the end-to-end performance of the design. Model checkers are traditionally optimized to answer yes/no type questions, and therefore require the performance estimation results as input parameters (*i.e.*, is the end-to-end computation time less than the estimate?). This provides a way to prove the schedulability and performance of the CMP design.

B. Relationship between Functional and Real-time Analysis

CARTA proposes the use of cycle-accurate FSM models for functional verification, and transaction-level DE/TA models for real-time analysis. While the transaction-level models are more abstract, the FSM models are not a direct refinement of either the DE or TA models. Therefore, it is important to clarify how the functional analysis relates to real-time analysis in the proposed approach.

The DES-based performance estimation and the timed automata-based real-time verification are based on an event-driven communication paradigm, where it is assumed that messages are passed through the interconnect as atomic transactions. In most CMP interconnect protocols – including the AMBA AHB on-chip communication architecture – hundreds of signals can transition during a data read or write issued from a processor to a memory. The DE and timed automata models abstract out this complexity to reduce modeling time and improve simulation speed, while still maintaining cycle accuracy required for meaningful exploration. Communication on the bus is represented as simple event passing between masters and slaves.

For *functional verification*, it is important to accurately capture the signals in the bus matrix interconnection network. If the analysis model is too abstract, certain problems can remain undetected in the design phase. Therefore, it is necessary to consider the actual signals that pass on the interconnect for functional verification.

Creating a cycle-accurate representation of the whole CMP application, however, is not necessary for practical analysis. The question that functional

verification aims to address is to show that there cannot be a combination or sequence of signals on the CMP interconnect that leads to a deadlock or livelock. We also show that masters get access to the interconnect whenever they request access, and can safely perform read and write operations. Once these conditions are shown, there is no need to consider cycle-accurate simulations of the actual CMP application for functional verification purposes.

The functional analysis has to capture the communication architecture of the CMP design, arbitration policies, masters/slaves connected to the bus, and focus on the various combinations and sequences of signals that are allowed by the protocol. This analysis does not consider the behavior of individual components for the functional verification, but rather focuses on problems that may arise when designers integrate components by industry standard interconnect protocols. If there cannot be any sequence or combination of bus signals that leads to deadlocks or livelocks, we show that the CMP design is functionally correct, and can build on this result to utilize transaction-level models for real-time analysis.

The DES-based performance estimation method and the timed automata-based analysis builds on the results of functional verification, as we can safely assume that masters and slaves can reliably communicate over the interconnect, following an event-based message passing paradigm. Thus we get the best of both worlds; we can prove functional correctness, and also benefit from improved analysis performance in transaction-level models.

III. Networking Router CMP Design

To demonstrate the effectiveness of our design flow, we use CARTA to explore a networking router case study. This design is a high-level abstraction of a router design by Conexant Systems, that was first described in [17]. This system is used for data packet forwarding, processing and encoding. The CMP design implementation for this case study consists of multiple processors, memories and network interfaces, and a bus matrix interconnection network. The different application tasks are mapped onto the CMP components shown in Fig. 3. Fig. 3 shows a simplified version of the hardware platform, without peripherals (e.g. timer, interrupt controller, UART, etc.) and without the DMA engine, for

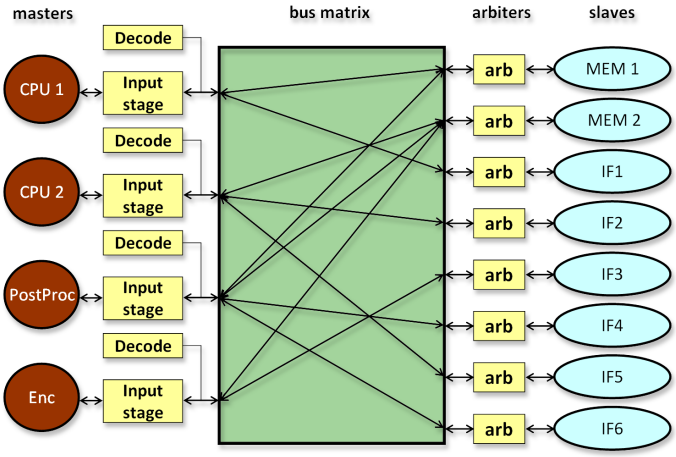

Fig. 3. Networking Router CMP HW Design

clarity. The major components in the CMP are the two embedded processors (`CPU_1` and `CPU_2`), a post processing dedicated HW (`PostProc`), an encrypt engine dedicated HW (`Enc`), on-chip memories (`MEM 1` and `MEM 2`), network interfaces to communicate with external components(`IF1 to IF6`), and an AMBA AHB bus matrix interconnection network (Multi-layer AHB) to facilitate inter-component communication on the CMP.

Several terms are used in the research community when referring to interconnect architectures. Terms such as bus matrix, crossbar switch, and multi-layer all refer to interconnect designs where masters and slaves are connected by more than one bus. Buses in bus matrix interconnects are also referred to as "layers", and buses containing a single master and slave are also referred to as "links".

The AMBA AHB protocol allows the creation of buses where only a single master and slave is connected. In this paper we use the term "bus" in this sense, to emphasize the fact that all masters and slaves communicate through standard AMBA AHB interfaces instead of being directly linked to each other by wires. We use the term "bus matrix" to refer to the terms Multi-layer AHB and AHB-Lite by ARM. AHB-Lite specifically considers the special case where only a single bus master is used for each bus (layer).

There are several advantages to using an industry standard bus between a single master and slave. Most importantly, it allows the use of any component in the design that supports the bus protocol standard. In the case of AMBA AHB, designers

can pick any AMBA AHB master and slave from third parties, and know that they can be reliably connected by AMBA AHB. Other advantages are that it provides a way to request retransmission of data from the master in the case of errors (i.e. RETRY response), it can be used to slow down the master if the slave needs more time to process requests (i.e. HREADY), and more importantly, it is much easier to extend in the future (by adding more masters/slaves) than a direct wire between a master and slave. On the slave side, generally a point arbiter (slave multiplexor) is used. This point arbiter can control the HREADY signal on each bus connected to the slave. Therefore, it can pick which master gets access to the slave by setting HREADY for that bus, and disabling for others. AHB-Lite is marketed by ARM for bus matrix interconnects where only a single bus masters are used. AHB-Lite removes the request/grant protocol and does not support RETRY/SPLIT from slaves. However, it also does not support slaves that may issue RETRY/SPLIT responses.

Fig. 4 shows a high level overview of the software design modeled as a task graph, as well as the mapping of the different tasks onto the CMP components. Computation tasks `(T)` mapped to the same processing unit communicate directly with each other. The blocks marked as `(B)` represent accesses to the bus matrix, either to read/write one of the memory units `(M)`, or to communicate with the external environment through network interfaces `(IF)`. Processor `CPU_1` is used to execute tasks associated with the intrusion detection functionality, while processor `CPU_2` executes tasks associated with the simple protocol translation and packet forwarding functionalities. The encrypt engine dedicated HW block is optimized for data encoding, and executes tasks that perform different encodings on packet data. A post-processing dedicated HW block is used to speed up the back-end of the protocol translation, as well as the encoding functionalities.

We now describe the networking router application case study in more detail. The system receives data packets from the network interface `(IF)` components. The receiver `(PktRx)` tasks are responsible for data packet pre-processing and preliminary decoding. Subsequently, the system performs multiple functions - intrusion detection, simple protocol translation, forwarding, and packet encoding. The intrusion detection functionality consists of a `Chk` task that is used to check if the packets have not been subjected to some suspicious activity, as would be the case if there is an intrusion. The `HdrCal` task is used to perform data packet processing to detect intrusions, based on user-defined intrusion signatures. If an intrusion is detected, then reports of suspicious activities are generated. Finally the packets are stored in physical memory `Mem_1` by the `Mem1` task from where another subsystem either blocks the flow, or passes the packets on to the next router. In the simple protocol translation functionality, the `Proc1` task is used to strip the source protocol and store the data packets into physical memory `Mem_1` by task `Mem2`. The memory also consists of a set of translational templates. These templates are used by the `Cnfg` task to strip the source protocol header, and then append the new protocol information to the data. The `Recalc_1` task finally reorders the data payload and new header, and sends out the packets to an outgoing network interface. The `Assm` task is used instead of the `Cnfg` and `Recalc_1` tasks if the protocol translation is fairly lightweight (for instance if the source and destination protocols are similar). The forwarding functionality consists of a task `Pkt_fwd` which receives a packet and updates the header data (e.g., updating time stamps and stripping source routing fields), and then forwards the data to the output interface. `Enc` implements the packet data encoding functionality, and stores the results into physical memory `Mem_2` by task `Mem3`. Subsequently, task `Recalc_2` is responsi-

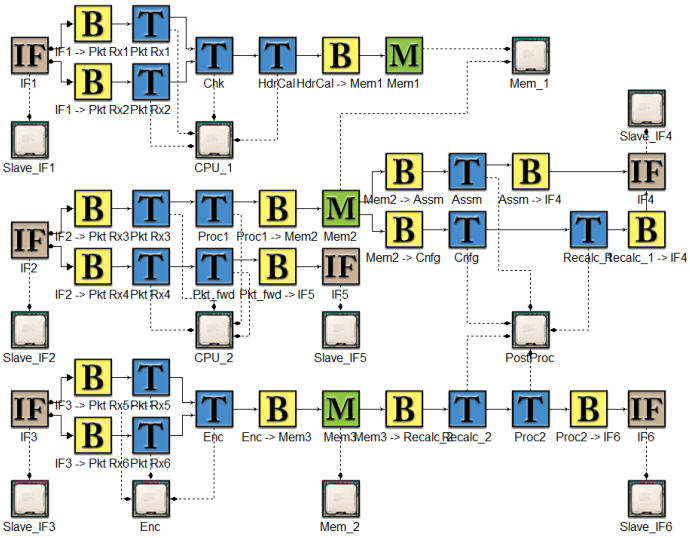

Fig. 4. Networking Router CMP SW Design

ble for reordering the data payload and creating a header. Task `Proc2` is used to finally perform post-processing the packet data headers before forwarding them to the output network interface.

IV. Modeling Bus Matrix-based CMP Designs

A. The ALDERIS Domain-Specific Modeling Language

There are three basic constructs in ALDERIS; *timers*, *tasks*, and *channels*. Tasks represent computations or resource utilization in CMP systems, and are characterized by an execution interval. Channels are simple FIFO buffers that can also express communication delays. Timers are special tasks that periodically broadcast events, triggering the execution of its dependents. These three constructs allow modeling of CMP designs as a set of interconnected task graphs.

The tasks and timers in ALDERIS are mapped to *threads*. Threads represent non-preemptive schedulers – tasks and timers mapped to a thread are scheduled by a fixed-priority non-preemptive scheduler. Threads are in turn mapped to *CPUs*. ALDERIS makes use of *components* to express hierarchy. Components may contain tasks and channels, and communicate through input/output *ports*.The ALDERIS DSML is freely available for download at http://alderis.ics.uci.edu.

We refer to the semantic domain of the ALDERIS DSML as the DRE model of computation (MoC). The semantics of the DRE MoC were formally specified as a discrete event system [3], and as network of timed automata [4], [5], [1], [6], [7].

A major advantage of having both a *discrete event (DE)* and *timed automata (TA)* representation is that we can choose which one we prefer to use depending on the problem that needs to be addressed and the required abstraction level. In this paper, we use the DE formalism for performance estimation, and the TA representation for real-time verification.

B. Modeling the Router CMP using ALDERIS

The networking router design shown in Fig. 3 uses an AMBA AHB bus matrix interconnection network. This architecture simplifies the arbitration policy – simple point arbitration is used at the slaves (either memories or network interfaces) to determine which master gets access to the slave. If the slave is not already busy serving a master, any master can get access to it immediately, and transmit data through its dedicated bus to the slave. If the slave is already busy serving a request, then the masters trying to access the slave are forced to wait. When the slave is free again, the master with the highest priority gets access to the slave. This simple point arbitration policy provides a great fit for simple task scheduling problems – the memory is modeled as a task, and scheduling this task for execution represents the access of the memory by other tasks. This abstraction provides a simple, but accurate model to capture the event flow between tasks, as well as memory and interface accesses through the bus matrix.

In the following we describe how we used the ALDERIS *domain-specific modeling language (DSML)* to model the case study described in Section III. Tasks and interfaces are modeled as ALDERIS tasks. We introduce FIFO buffers between tasks in order to (1) capture communication delays on the bus, and (2) buffer events, in case the task is already executing, and not able to receive the event yet. Dependencies in the ALDERIS task graph follow the SW design dependencies. However, there are some differences, as the ALDERIS models capture concurrency, real-time properties, and scheduling as well. We add timers to the model that represent the sampling rates over the input interfaces. Timers periodically push events, that represent data received from the environment. The task graph is event-driven and asynchronous, therefore further event propagation is not synchronized, timers are solely used as a triggering mechanism.

Fig. 4 shows the dependencies in the SW design. It does not, however, capture the semantics of the event flow. There are four branches in the model (from `IF1`, `IF2`, `IF3`, and `Mem2`). The branches from `IF1`, `IF2`, and `IF3` represent a choice where only one path is taken. The branch from `Mem2` to `Assm` and `Cnfg` represent broadcast – both dependents receive the event, and process it when they are scheduled for execution.

One physical HW unit may be represented as not just one, but several `CPU` constructs in ALDERIS. When a choice is made, a simulation trace will choose either one of the paths depending on certain conditions. However, to calculate the worst-case performance, the formal analysis has to capture both

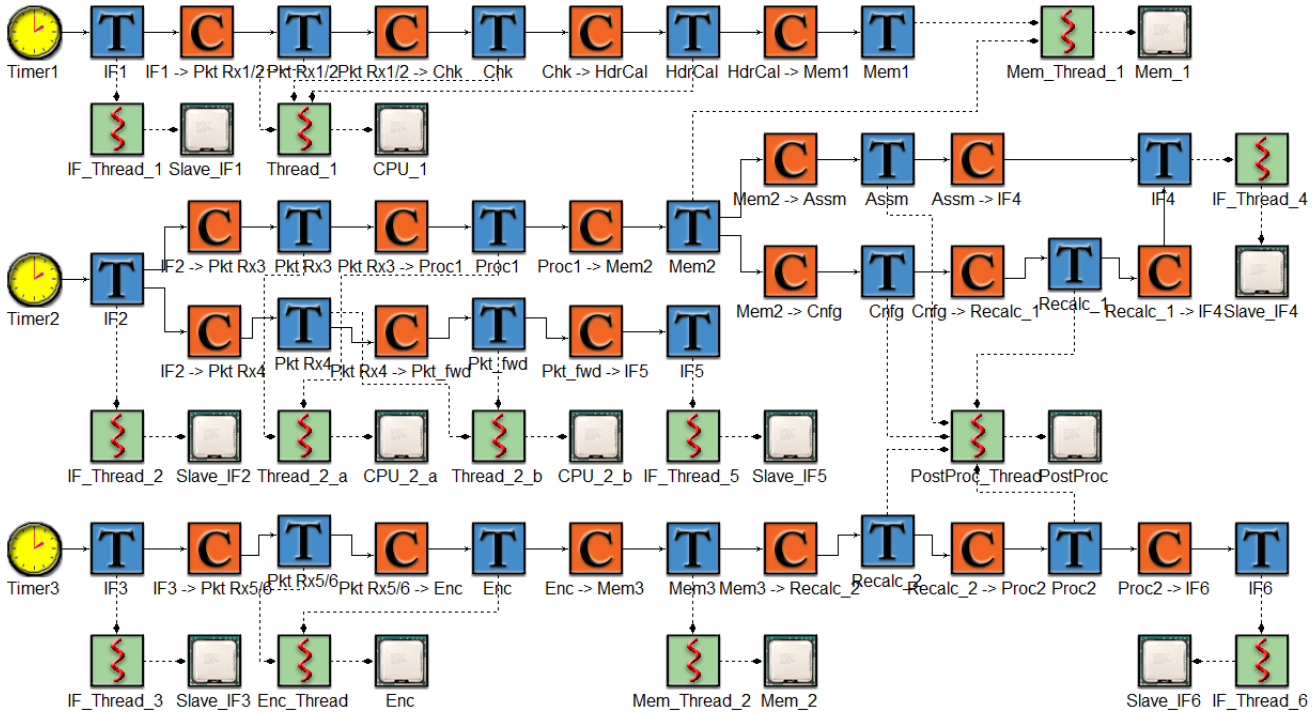

Fig. 5. ALDERIS Model of the Router CMP in the GME Tool

options, so we build a model where both paths are simultaneously taken. The `CPU_2` HW component shown in Fig. 4 is modeled as two independent (logical) `CPUs` in the ALDERIS model shown in Fig. 5 (`CPU_2_a` and `CPU_2_b`). When running a simulation, only one path is active, and this path is running on the physical `CPU_2` HW. When we perform the formal exploration, we duplicate the `CPU_2` physical processor to the `CPU_2_a` and `CPU_2_b` logical processors, and execute both paths simultaneously on these independent `CPU` models. This approach is feasible as the execution of a task on one path does not interfere with the execution of another task from the other path. Therefore, the formal model concurrently explores both possible traces simultaneously.

`IF2` is the interface for the protocol translation functionality. Depending on whether the data is forwarded or stored in the memory for further processing, one of two distinct event paths is taken from `IF2`. Although both paths execute on the same processing unit `CPU_2`, these two paths cannot be active at the same time; either the `PktRx3`, `Proc1` path is taken (and then `Mem2` and so on), or the `PktRx4`, `Pkt_fwd`, `IF5` path. Even though these paths are mapped to the same processing unit, they are not concurrent. Therefore, we execute both paths in parallel for performance estimation. By introducing a new logical processing unit, `CPU_2_b`, we can simultaneously explore both paths at the same time.

We encounter the same situation at the branch from `IF1`; either `PktRx1`, or `PktRx2` is chosen to process the packages arriving through the interface. Therefore, we have the option to introduce a new logical processing unit, to capture the fact that `PktRx1` and `PktRx2` do not execute at the same time. Multiple logical masters are generally required anytime a choice is made. However, if the choice is performed within a single thread, and both paths merge before any task communicates with tasks mapped to remote threads, the different paths may be grouped into a single task. This is a manual optimization technique that may improve analysis performance.

We utilize the manual optimization in this case, as both `PktRx1` and `PktRx2` are mapped to the same thread, and both paths merge before the bus access is made. We substitute a new logical task `(PktRx1/2)`, with a *best case execution time* (bcet) of min[bcet(`PktRx1`), bcet(`PktRx2`)], and a *worst case execution time* (wcet) of max[wcet(`PktRx1`), wcet(`PktRx2`)]. Regardless of which execution path is taken

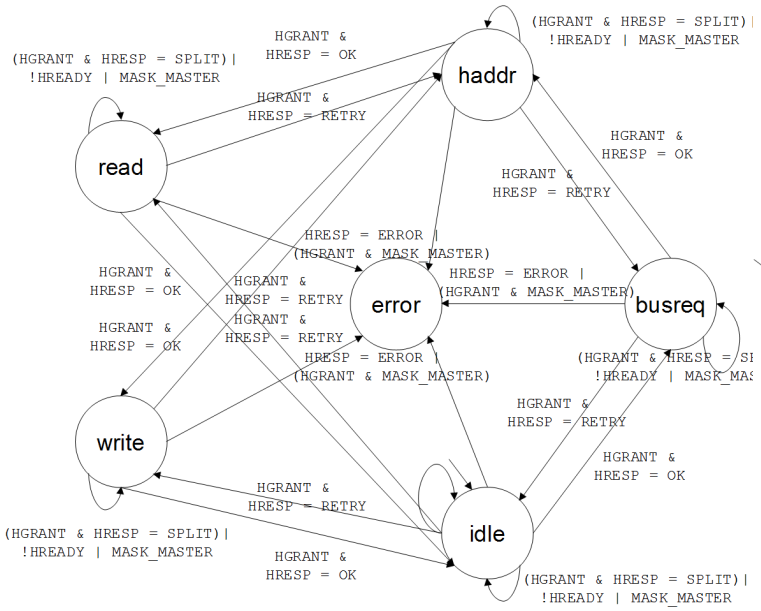

Fig. 6. Finite State Machine Model of an AMBA AHB Master

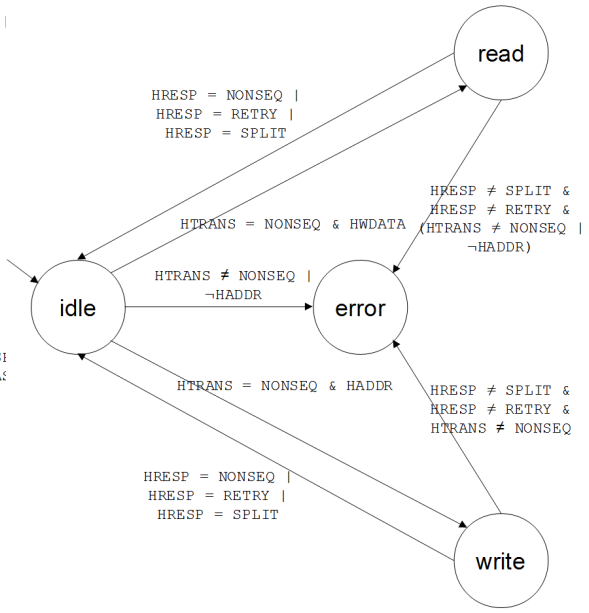

Fig. 7. Finite State Machine Model of an AMBA AHB Slave

(`PktRx1` or `PktRx2`), task `PktRx1/2` captures the execution intervals of both tasks, and therefore can be used for worst case performance estimation. This method increases scalability with minimal loss of accuracy, and is therefore preferable in the early stages of the design flow. We apply the same idea for the branch from `IF3`, and substitute tasks `PktRx5` and `PktRx6` with the `PktRx5/6` logical task. Finally, the branch from `Mem2` follows broadcast semantics to dependents, and therefore requires no changes in the ALDERIS representation. The resulting ALDERIS model shown in Fig. 5 captures dependencies between tasks, the mapping of tasks to the target platform, as well as timing information of the networking router CMP design, and allows formal analysis as described in the following sections.

V. Functional Verification of Bus Matrix CMP Designs

Functional verification is a key challenge in the design of complex CMP systems. Modern bus protocols are increasingly complex, and do not guarantee the correctness of the overall CMP design. Moreover, they are often specified by natural languages, and therefore there may remain ambiguities in the specification. In [2], we presented a method for the modeling and functional verification of CMP designs utilizing an AMBA AHB bus. In this paper, we apply this work to CMP designs using fully-connected AMBA AHB bus matrix interconnects.

We have considered two key problems in our formal analysis. A *deadlock* can be observed in the finite state machine model as a state where no transitions are enabled. A *livelock* can be observed as a state from which only a subset of states is reachable. A livelock can express situations like starvation, where a master is not granted access to the requested slave.

In a fully connected bus matrix, there is a dedicated AMBA AHB bus between each master and slave. Point arbitration is used to manage access to slaves (such as memories). The bus matrix design increases throughput and concurrency, as multiple transactions can take place simultaneously between different masters/slaves.

In bus matrix designs, the functional verification is simplified. Each bus in the bus matrix connects a single master to a single slave. Therefore, congestions on the bus are greatly reduced, and can only appear when the slave is busy serving another master. There are altogether 11 buses in the networking router case study, as shown in Fig. 3. Point arbitration is managed by a simple fixed-priority arbiter. Priorities for masters are defined as follows (in decreasing order): `CPU 1`, `CPU 2`, `PostProc`, `Enc`.

We have created a cycle-accurate model of the AMBA AHB bus protocol in order to model transactions over the bus accurately. We capture multiple

signals over the bus, such as BUSREQ, HREADY, HGRANT, HADDR, HTRANS, and HRESP. We consider RETRY responses from the slave (HRESP = RETRY), as well as split transfers, as we analyze Multi-layer AHB. AHB-Lite does not support split transfers, RETRY responses, or the request/grant protocol, and is therefore even simpler to analyze than Multi-layer AHB.

We did not model HLOCK signals for the analysis. HLOCK signals are set by a master that needs uninterrupted access to the bus during a transaction. If the master asserts HLOCK, the arbiter simply holds its state. A master that does not deassert HLOCK could cause a livelock by disallowing the slave to serve other masters. However, it is the responsibility of the master to manage the HLOCK signal, and a master that does not deassert HLOCK would be considered faulty. When the master deasserts HLOCK, the arbiter is free to continue where it left off, so no livelocks can occur.

AMBA AHB masters and slaves are modeled as abstract state machines as shown in Fig. 6 and Fig. 7. The models were manually created in accordance with the AMBA AHB specification. Masters are modeled using 6 states (idle, busreq, haddr, read, write, error), and slaves are modeled using 4 states (idle, write, read, error). We have modeled arbitration delays, busy slaves, and two-cycle response times for RETRY and SPLIT responses as defined in the AMBA AHB specification. SPLIT responses are initiated by the slave in the proposed FSM models non-deterministically.

We have used the NuSMV [11] model checker to verify CTL [18] properties on the networking router CMP design. We specified AMBA AHB masters, slaves, and the arbiter in NuSMV code. Algorithm 1 shows the NuSMV specification for an AMBA AHB arbiter managing a single master and slave. This model is sufficient to verify the correctness of the arbiter, as in a fully connected bus matrix each bus connects a single master to a single slave. Point arbitration is managed at the slave side; whenever the slave is busy serving a transaction, it signals this fact by setting the HREADY signal to low (other implementations are also possible). NuSMV models for AMBA AHB buses containing multiple masters using a round-robin arbiter are available at http://alderis.ics.uci.edu/amba2.html.

In [2] we have described an ambiguity in the final version of the AMBA AHB specification, that may arise when a slave splits a a master, and requests retransmission by setting the HRESP = RETRY response in the same bus cycle.

In a fully connected bus matrix, each AMBA AHB bus connects a single master and slave. Therefore, the slave has little reason to split the master. AHB-Lite does not support split transfers and RETRY responses, therefore the problem does not occur in AHB-Lite. The ambiguity, however, it applicable to Multi-layer AHB, as it supports both split transfers and RETRY responses. Therefore, the ambiguity is applicable to the networking router

Algorithm 1 NuSMV Specification of an AMBA AHB Arbiter Managing a Single Master and Slave

```
MODULE bus_matrix_arbiter (HTRANS, HREADY, HRESP,
BUSREQ, HGRANT, HMASTER, HSPLIT)
   VAR
    master_state : idle, mask, grant, transmit;
    master_prev_state : idle, mask, grant, transmit;
    lasterror : boolean;
    preferred : master, default;
   ASSIGN
    init (master_state) := idle;
    init (master_prev_state) := master_state;
    init (lasterror) := 0;
    init (preferred) := default;
    next (master_prev_state) := master_state;
    next (preferred) :=
      case
        -- Master starts the transmission
        !HGRANT & master_state != mask & HREADY &
HRESP = OK & BUSREQ : master;
        -- Split master
        HGRANT & HRESP = SPLIT : default;
        -- Master finishes the transaction
        HGRANT & master_state = transmit & HTRANS =
IDLE & HREADY & HRESP = OK & !BUSREQ :
          default;
        -- Master gives up its BUSREQ
        HGRANT & master_state = idle & HTRANS = IDLE
& HREADY & HRESP = OK & !BUSREQ : default;
        -- Unmasking masters
        !HGRANT & master_state = mask & HSPLIT =
master : master;
        1 : preferred;
      esac;
    next (master_state) :=
      case
        -- Roll back for retrys
        HRESP = RETRY & !lasterror :
master_prev_state;
        master_state = idle & HREADY & HRESP = OK &
HGRANT : grant;
        master_state = grant & HTRANS != IDLE &
HREADY & HRESP = OK & HGRANT : transmit;
        master_state = transmit & HTRANS = IDLE &
HREADY & HRESP = OK & HGRANT : idle;
        HREADY & HRESP = SPLIT & HGRANT & !lasterror
: mask;
        master_state = mask & HSPLIT = master : idle;
        lasterror : master_state;
        !HREADY : master_state;
        1 : master_state;
      esac;
    next (lasterror) :=
      case
        HRESP = RETRY : 1;
        HRESP = SPLIT : 1;
        1 : 0;
      esac;
    HMASTER :=
      case
        preferred = master : master;
        preferred = default : default;
      esac;
    HGRANT :=
      case
        preferred = master : 1;
        1 : 0;
      esac;
```

case study as well. We use the simple fix of disallowing the slave to split a master and set the HRESP signal to RETRY in the same cycle.

To show that no deadlocks can occur in the model, we have shown that the error state is unreachable in masters and slaves by checking the following CTL formulas (where x refers to the index of masters, y is the index of slaves):

```
AG (MASTERx.state != error),
AG (SLAVEy.state != error).
```

We have specified rules to enforce that whenever a master is split, it will be eventually unsplit in order to avoid livelocks. We have verified this property using the following CTL formulas:

```
AG ((MASK_MASTERx) ->
  AF (!MASK_MASTERx)).
```

Finally, we checked whether starvations are possible by checking the following formulas:

```
AG (MASTERx.state = busreq ->
        AF HGRANTx),
AG (MASTERx.state = busreq ->
  AF MASTERx.state = write).
```

A. Experiments

We have run experiments using the NuSMV tool on an Intel Core i7 processor running at 4GHz with 6GB triple-channel DDR3 RAM. For the arbiter connecting a single master and slave, the verification time took less than a second, with 6700KB memory consumption. For 2 masters, the analysis took less than a second with 11700K memory consumption. For 3 masters the analysis took 28 seconds with 92080KB memory consumption. Since in this paper we consider fully connected bus matrices only, scalability issues do not arise, as each master is connected to each slave by a dedicated AMBA AHB bus.

We have found that starvations are possible in general when high-priority masters continually request access to slaves, and therefore lower-priority masters do not get served. Therefore, we need to consider the actual dependencies in the model, and perform real-time analysis to ensure that this condition does not occur. We need to consider the actual communication between tasks in the CMP design, and consider whether the starvation may occur in the actual CMP design.

This problem, however, cannot be adequately addressed at the cycle accurate abstraction due to the long computation times, that would lead to state space explosion. There is no theoretical limitation on performing real-time verification at the cycle accurate abstraction; the limitiation is present simply due to the state space explosion present at low-level abstractions.

In the next section we describe how we obtained worst case performance estimates on the networking router case study, and how we used the results in the final stage for real-time verification using timed automata in Section VII. We address the problem of starvation in Section VII.

VI. Formal Performance Estimation by Discrete Event Simulations

This section describes how we utilized the open-source DREAM tool – that implements the DES-based simulation-guided performance estimation method – for the performance estimation of the networking router CMP design. We build on the fact that the design is based on a fully connected AMBA AHB bus matrix. When applying the proposed method to bus matrix designs that are not fully connected, accuracy loss is expected, as the DES-based performance estimation does not capture congestions on the buses due to arbitration policies. The CMP design shown in Fig. 5, however, utilizes a fully connected bus matrix as shown in Fig. 3, therefore arbitration policies can be captured as point arbitration at the slave side.

Table I shows the parameters used for the model shown in Fig. 5. We have obtained the deadlines from application requirements, and the execution time estimates by using fast and accurate system-level simulation at the *Cycle Count Accurate At Transaction Boundaries (CCATB)* abstraction [16].

Fig. 8 illustrates the DES-based performance estimation method. The dependencies imply a partial ordering on the execution order of tasks, as well as the events during the discrete event simulation. We distinguish three major types of events in Fig. 8; *input* events, denoted as "i", *output* events denoted as "o" and *start* events denoted as "s". Each event is followed by a number to indicate which task is responsible for the event. This is simply a syntactic short-hand to keep the nodes in the tree small. Numberings for tasks are given in the last column of Table I.

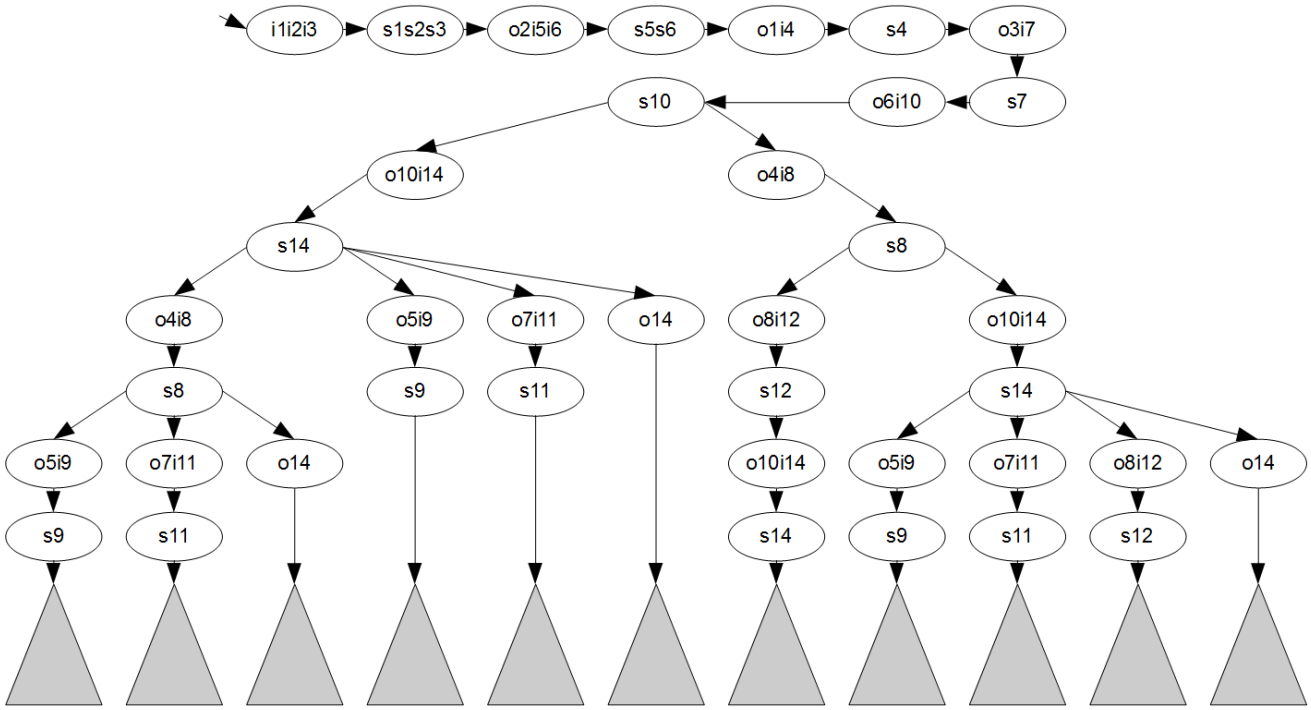

Fig. 8. A Partial View of the Event Order Tree for the Example Shown in Fig. 5 using the Parameters in Table I

Task	Priority	WCET	BCET	DL	No
IF1	1	200	-	201	1
IF2	2	100	-	101	2
IF3	3	400	-	401	3
IF4	4	200	-	201	22
IF5	5	200	-	201	14
IF6	6	400	-	401	23
Mem1	7	100	-	201	16
Mem2	8	100	-	201	13
Mem3	9	100	-	101	15
PktRx1/2	10	1442	770	1443	4
Chk	11	450	220	451	8
HdrCal	12	2400	1340	2401	12
PktRx3	13	1530	1400	1531	5
PktRx4	14	670	590	671	6
Pkt_fwd	15	600	200	601	10
Proc1	16	2800	2400	2801	9
Cnfg	17	1600	1300	1601	18
Assm	18	800	600	2401	17
Recalc_1	19	3000	2000	3801	20
PktRx5/6	20	2100	1100	2101	7
Enc	21	4900	3800	4901	11
Recalc_2	22	1750	1500	6381	19
Proc2	23	5600	4800	5601	21

TABLE I
PARAMETERS FOR THE NETWORKING ROUTER CMP DESIGN SHOWN IN FIG. 5

As seen from Fig. 8, the application starts with tasks IF1, IF2, and IF3 receiving input events. The second node shows that the three tasks are scheduled (started) by the scheduler for execution. Since IF2 has the smallest execution time between IF1, IF2, and IF3, therefore IF2 will be the first to finish its execution. When IF2 finishes its execution, it generates an output event, that is broadcasted to both PktRx3 and PktRx4.

The first non-deterministic branch occurs after Pkt_fwd (numbered 10) is scheduled for execution. The earliest time when Pkt_fwd may finish its execution is at time 890, as can be computed by adding the best case execution times of itself and its sources IF2, PktRx4 (100+590+200=890). We can also compute that Pkt_fwd will finish its execution by time 1370 (100+670+600=1370). For task PktRx1/2 we can similarly obtain that it will finish its execution between 970 and 1642. Therefore, the execution intervals of Pkt_fwd and PktRx1/2 overlap, and we need to consider two cases; when Pkt_fwd finishes first, and when PktRx1/2 finishes first. It is also possible that the two tasks finish simultaneously, in which case race conditions may be considered, resulting in the same two options. Total orderings become important when multiple tasks are mapped to the same thread

Algorithm 2 Obtaining and Enumerating the Event Order Tree by Discrete Event Simulations

1: create the (empty) superset of race conditions R
2: set the execution time for all tasks $t_k \in T$ to their wcet_k time, and the next execution time for all tasks to their bcet_k time, respectively ($\forall t_k \in T$ $\mathsf{exec\_time}_k = \mathsf{wcet}_k$, $\mathsf{next\_time}_k = \mathsf{bcet}_k$)
3: *// enumerate all branching intervals*
4: **for all** permutations of $\mathsf{exec\_time}_k$ assignments, obtained using the $\mathsf{next\_time}_k$ variables **do**
5: clear the superset R
6: **call** discrete_event_simulation () described in Algorithm 3
7: *// enumerate all race conditions with the current* $\mathsf{exec\_time}_k$ *assignments*
8: **for all** permutations of events in superset R **do**
9: **call** discrete_event_simulation () described in Algorithm 3
10: **end for**
11: **end for**

Algorithm 3 function discrete_event_simulation ()

1: run directed discrete event simulation, during which each task stores its start time as start_k
2: during the simulation all tasks t_k observe events e_i that are raised in the [start_k + bcet_k, start_k + $\mathsf{exec\_time}_k$] interval
3: **if** start_k + $\mathsf{next\_time}_k$ < $\mathsf{time}(e_i)$ **then**
4: *// we have encountered a branching point in the (*bcet_k, wcet_k*) interval*
5: store the value of $\mathsf{time}(e_i)$ - start_k in the $\mathsf{next\_time}_k$ variable
6: **else**
7: do nothing, event will be considered in subsequent simulations
8: **end if**
9: *// find all race conditions with the current* $\mathsf{exec\_time}_k$ *assignments*
10: **for all** race conditions detected between events $e_i, e_j, \ldots e_k$ during the simulation **do**
11: search for the set containing events $e_i, e_j, \ldots e_k$ in superset R (from Algorithm 2)
12: **if** the set is found **then**
13: do nothing
14: **else**
15: add the set $S = \{e_i, e_j, \ldots e_k\}$ to the superset R
16: **end if**
17: **end for**

or CPU. For example, both `Mem1` and `Mem2` denote memory accesses in the same memory module. Likewise, tasks `Cnfg`, `Recalc_1`, `Recalc_2` and `Proc2` are all mapped to the same thread, and therefore it is important to consider the order in which they may be scheduled for execution. Fig. 8 only illustrates the top of the tree, and the grey rectangles refer to subtrees of the corresponding nodes.

The size of the event order tree grows very fast even for moderate-size examples, and precomputing the tree is not possible due to resource constraints. Rather, the proposed method builds the event order tree on-the-fly, that captures all the possible total orderings of events. Branches are discovered during simulation-time, and then subsequently enumerated. Algorithm 2 describes how the event order tree is constructed on-the-fly.

There may be potentially exponential numbers of paths in the analysis. This is not the characteristic of the analysis method, rather that of the analyzed model. When non-determinism is present in the models, all possible paths have to be considered to guarantee that all scenarios that were considered for the performance analysis. CARTA can express deterministic models, or time-triggered models as well, in which case the analysis is polynomial. A key advantage of the proposed DES-based performance estimation approach is that it does not restrict designers' hands, the bound on analysis performance strictly depends on computation power, not arbitrary restrictions on the models.

By enumerating the event order tree, one can obtain the worst case bound on the overall performance of the model. To enumerate the tree, the discrete event simulation step described in 3 are performed repeatedly, where the execution time of tasks is continuously updated to capture all possible permutations of execution times. Since the analysis is based on repeated simulations, we refer to this approach as "simulation-guided model checking". The event order tree captures all permutations of events, and is therefore exhaustive; it does not produce false positives. On the other hand, the method is built on the assumption that discrete event simulations using limited horizon are sufficient, after which the system returns to an initial idle state, which may not be the case for all types of real-time systems. For a description of the formal performance estimation method and proofs on the validity of the performance estimation method please see [3].

The DES-based method is more accurate for performance estimation than static analysis methods, as

it captures dynamic effects such as congestions on the bus, and race conditions. The advantage of the DES-based method compared to ad-hoc simulations is the increased state space coverage. Compared to pure model checking methods, the main advantage is that it does not run out of memory on large-scale examples, as it is based on fast iterative simulations, and is therefore CPU-bound. Moreover, most model checkers are tailored to answer yes/no questions, but the DES-based method can directly obtain the worst case bound on the end-to-end computations.

A. Experiments

The open-source DREAM tool computed the worst case end-to-end performance of the networking CMP design modeled as shown in Fig. 5 in 590 seconds on an Intel Core i7 920 processor running at 4GHz using 6GB triple channel memory, with only 776KB memory consumption. The implementation of the DES-based method was not optimized to take advantage of the multi-core architecture, and therefore executed on a single thread. The DES-based performance estimation method is easy to parallelize as it consists of repetitive simulations, and we are considering a multi-core implementation in the future.

The overall end-to-end performance estimate is 17780 cycles. We use this information as a bound on the end-to-end performance of the networking router CMP design. Each cycle in the model represents $5ns$. The lowest period of the timers that still does not violate the end-to-end bound is therefore $\frac{17780\times5}{10^3} = 88.9\mu s$. This means that the highest possible frequency for the sampling is $\sim$11.248KHz. Note that with each sampling the networking router CMP processes several packets, and therefore provides reasonable performance. The analysis is exhaustive, and therefore the bound is tight. In the next section we perform real-time verification on the model to prove that the performance estimates hold, and that no starvation occurs in the CMP design.

VII. REAL-TIME VERIFICATION USING TIMED AUTOMATA

This section describes the proposed approach for real-time verification by timed automata. This method also assumes a fully connected bus matrix for the analysis, as arbitration policies are not captured in the formal models. When applying the proposed method to bus matrix designs that are not fully connected, accuracy loss is expected. The CMP design shown in Fig. 5, however, utilizes a fully connected bus matrix as shown in Fig. 3, therefore arbitration policies can be captured as point arbitration at the slave side.

Real-time verification is optional in some cases, as the performance estimation method is based on an exhaustive state space search, and is therefore a model checking method itself. There are three cases when the use of the extra verification step is justified. First, the *discrete event simulation (DES)*-based performance estimation method does not capture timed states to improve scalability, but rather utilizes a *limited horizon* for the simulations. Although this approach is sufficient in most cases where the simulation periodically returns to the initial state, it cannot be applied to all models in general. For example, obtaining the required time horizon for simulating a pipeline architecture may be error-prone. In other words, while the DES-based method does not produce false positives, it relies on the assumption that a limited horizon is sufficient for the analysis. In cases where this condition cannot be proven, the use of the additional verification step is required.

Second, for some complex CMP designs the performance estimation method *might not terminate* due to the state space explosion problem. TA-based model checkers in some cases (but not always) achieve better scalability. In this case, the use of the real-time verification method is justified, as it might prove the validity of the worst case performance estimate.

Third, the TA-based model checker can be utilized to *prove properties* on the design that could not be carried out at the cycle-accurate level due to the state space explosion problem. Our *finite state machine (FSM)*-based analysis described in Section V has shown that starvations may be present in the design due to the fixed-priority point arbitration algorithm utilized in the slaves. By capturing the CMP design shown in Fig. 5 as TA, we can prove that no livelocks and starvations are present in the model. Note that real-time properties such as execution times may influence starvations, and therefore have to be considered for the analysis.

In this paper we utilize the timed automata model checking due to the third condition; it is hard to

prove that no deadlocks and livelocks are present using the DES-based method. During DES, we can easily identify situations where a task did not execute at all, but it is hard to identify whether a blocking occurs simply due to waiting for resources, or an actual deadlock or livelock. The timed automata-based method removes any doubt, and there is no reason not to use it given that the models can be automatically generated from DREAM.

Fig. 9 shows the partial TA representation in the UPPAAL tool for the networking router CMP design shown in Fig. 5. The locations denoted with U are urgent locations, and C denotes committed locations [12], both of which imply that time cannot pass in that location, and the outgoing transitions needs to be taken immediately upon entering the location. Tasks have two clocks, c_e and c_d. Tasks, channels, timers, and schedulers compose together as a network of timed automata, providing an abstract model for scheduling. The translation from ALDERIS models to the TA representation is described in detail in [4], [5], [6], [7]. The translation process itself is based on refinement, DREAM generates a timed automata model for each task, FIFO buffer, and timer in the ALDERIS models using a template. Scheduling policies are specified as automata, where transitions may trigger the execution of tasks.

The translation is implemented in the open-source DREAM tool and is fully automated. DREAM generates timed automata representations for the UPPAAL and Verimag IF model checkers. In this section we arbitrarily focus on UPPAAL, as both tools are timed automata model checkers. We have verified that no task can violate its deadline by checking the following UPPAAL macro:

$$\texttt{A[] not deadlock} \tag{1}$$

Since the timed automata are created in such a way to deadlock when a deadline is violated, this analysis proves the real-time schedulability of the system. Note that this result does not contradict the FSM-based analysis. The FSM-based analysis shows that deadlocks may be present in general in CMP designs utilizing AMBA AHB with fixed-priority point arbitration. When we consider the actual communication in the CMP design by considering dependencies and when components request access to resources in the timed automata model, we can prove that no deadlock are present in the actual CMP design. In short, deadlocks may be present in general, but are not present in the router case study used in this paper.

A. Experiments

We have run experiments using the UPPAAL model checker on an Intel Core i7 920 processor running at 4GHz using 6GB triple channel memory. The verification time took less than a second for the design illustrated in Fig. 9, with 9140KB memory consumption.

We have rounded up the period of the timer to 18000 cycles for simplicity, and we were able to prove the end-to-end execution time of 17780 cycles. The real-time verification shows that any frequency that is lower than the corresponding 11.1KHz is guaranteed not to violate the end-to-end deadline, or the individual deadlines of tasks.

VIII. COMPARING THE RESULTS OF THE ANALYSIS METHODS

Fig. 10 illustrates the analysis time and memory consumption used for the analysis of the networking router case study. During the functional verification phase, we considered a cycle-accurate model of the AMBA AHB bus, and found that deadlocks may be present due to the fixed-priority arbitration. The analysis time took less than one second for buses containing a single master and slave, however we observed increased analysis time and memory consumption as more and more masters were added to the bus. This was mainly the result of the more complex arbitration policy needed. Since in fully connected bus matrices each master is connected to each slave, analysis scalability was not an issue.

For the performance estimation, we relied on the DES-based simulation-guided model checking method. Analysis time was higher, while memory consumption was lower. Generally, we find that memory-bound model checkers have much better performance than the CPU-bound method if the example is actually small enough to fit in the main memory. Once the model size increases beyond the main memory size, memory-bound model checkers are not useful in practice, as performance degrades significantly, and no partial results are given. For this example, we find that the model is small enough to fit in memory, and the various optimizations

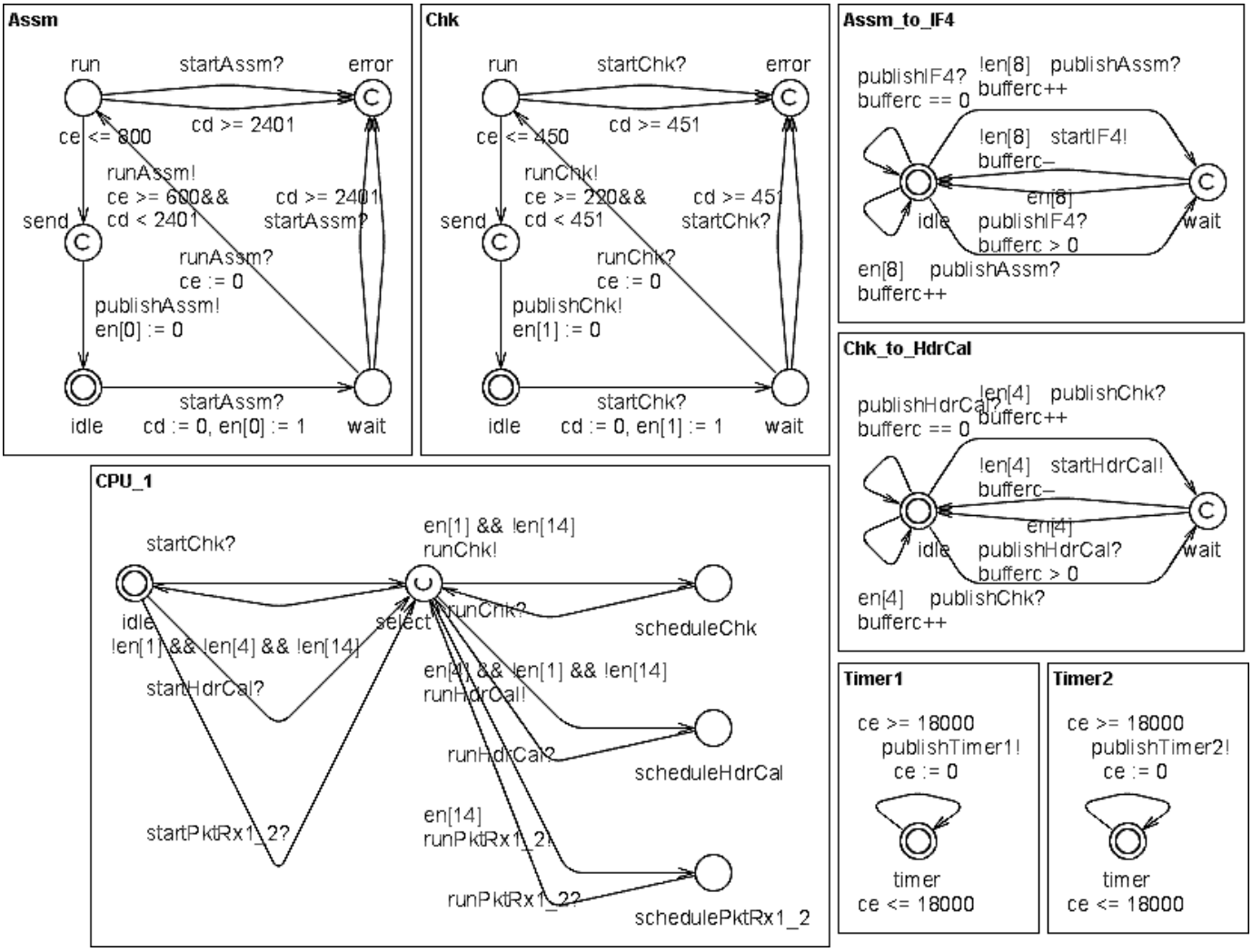

Fig. 9. Partial Timed Automata Model of the Networking Router CMP Design Shown in Fig. 5 in UPPAAL

result in much improved performance compared to the DES-based method. That said, 590 seconds is completely acceptable to obtain the worst case end-to-end bound on the networking router design.

For real-time verification, the UPPAAL tool shows impressive performance by proving deadlock-freedom, as well as proving the real-time schedulability of the router design. Results are similar to the NuSMV results, even though the problem analyzed is different: UPPAAL considers the interactions between tasks – similarly to the DES-based method – on a transaction-level abstraction. Here we see the advantage of using multiple abstractions for the analysis; performing the same analysis using the NuSMV tool at the cycle-accurate level would result in serious performance penalty, and possibly even state space explosion. Our results show the practical applicability of the CARTA framework for the cross-abstraction analysis of CMP designs.

IX. RELATED WORK

Model-based Design: Ptolemy II [19] is a modeling framework that composes heterogeneous models of computation, and performs simulation for symbolic performance estimation. Platform-based design was proposed for the system-level design of embedded systems [20], including wireless sensor networks [21]. A model-driven design framework for the development of embedded systems on GME is described in [10]. CARTA also utilizes the GME tool, but the focus is on formal analysis rather than development. CARTA also integrates multiple model checkers, and CCATB simulations for the comprehensive functional verification, performance estimation, and real-time verification of bus matrix CMP designs.

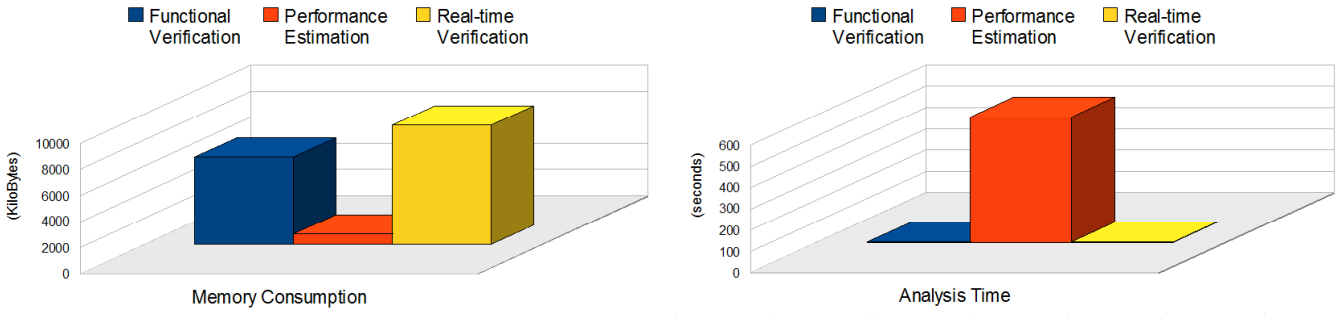

Fig. 10. Analysis Time and Memory Consumption for the Networking Router Case Study

We have introduced the ALDERIS DSML in [1]. ALDERIS models embedded systems as task graphs mapped to logical threads and processing units, and captures key execution times, delays, and deadlines of the design for formal analysis. We utilize the ALDERIS DSML in this paper as the high-level specification of the design, that drives the overall model-based design flow. The novelty in this paper is to show how the ALDERIS DSML can be applied for the analysis of CMPs utilizing complex bus matrix interconnects.

Functional verification of bus architectures: An early work on applying model checking methods to the AMBA protocol was presented in [22]. A verification platform for AMBA-ARM7 is presented in [23]. A verification platform for AMBA using a combination of model checking and theorem proving is described in [24].

All the works described above utilize formal methods to prove the functional correctness of bus *protocols*. In contrast, CARTA focuses on the analysis of CMP designs and aims to prove that no deadlocks and livelocks can occur in in a CMP design – a property not guaranteed by the AMBA AHB protocol itself. The novelty in this paper is to extend our earlier work on the functional verification and performance analysis of AMBA-based *CMP designs* described in [2] to complex AMBA AHB bus matrix architectures.

Performance analysis: The SymTA/S [25] approach proposed methods from scheduling theory for the performance analysis of embedded systems. A component-based framework for schedulability analysis and performance evaluation was proposed in [26]. Modular Performance Analysis [27] is an approach based on real-time calculus, that models dependencies as arrival curves.

Simulations are widely used methods for performance estimation by the industry. Disadvantages of these approaches include immeasurable state space coverage, and long development times resulting in performance issues being detected too late in the design cycle, implying higher costs to address changes. A semi-formal simulation-based performance estimation technique was proposed in [28].

We have presented a method for the formal performance estimation of *distributed real-time embedded (DRE)* systems using discrete event simulations in [3]. This approach is a model checking method based on iterative simulations, and is more accurate than static performance estimation methods, and improves on the coverage of ad-hoc simulation-based methods. Moreover, it is able to provide counter-examples when real-time properties are violated. In this paper we extend this method for the performance estimation of CMP designs, and show a novel method to abstract out bus matrices into the DES model used by the estimation algorithm.

Real-time verification: Model checking is an alternative approach for symbolic performance verification. Timed automata were proposed as model of computation for real-time verification of task graphs [29]. A timed automata-based approach for the thread-level analysis of DRE systems is presented in [30]. CADENA [31] is an analysis framework for building and analyzing *CORBA Component Model (CCM)*-based systems. This paper extends our earlier work on the real-time verification of DRE systems based on timed automata described in [4], [5], [6], [7]. We extend these works for the real-time verification of CMP designs utilizing bus matrices. Moreover, we show how the timed automata model checkers are integrated into our model-based CARTA design flow.

X. Conclusion

We have presented the *Cross-abstraction Real-time Analysis* (CARTA) framework for the model-based functional verification and performance estimation of *chip multiprocessors (CMP)* utilizing bus matrix (crossbar switch) interconnection networks. The CARTA design flow aims to address three major challenges in the formal analysis of CMP designs: *(1) functional verification*, *(2) performance estimation*, and *(3) real-time verification*, and provides a cross-abstraction bridge between the *finite state machine (FSM)*, *discrete event (DE)* and *timed automata (TA)* models of computation. CARTA utilizes multiple model checkers to analyze formal properties at the cycle-accurate and transaction-level abstractions. We have demonstrated the applicability, as well as the scalability of the prototype analysis methods implemented in the CARTA framework on a networking router CMP design. The ALDERIS modeling language (http://alderis.ics.uci.edu) and the open-source DREAM tool (http://dre.sourceforge.net) are free to use for any purpose.

References

[1] G. Madl and N. Dutt, "Domain-specific Modeling of Power Aware Distributed Real-time Embedded Systems," in *Proceedings of the 6th Workshop on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, 2006.

[2] G. Madl, S. Pasricha, Q. Zhu, L. A. D. Bathen, and N. Dutt, "Formal Performance Evaluation of AMBA-based System-on-Chip Designs," in *Proceedings of EMSOFT*, 2006, pp. 311–320.

[3] G. Madl, N. Dutt, and S. Abdelwahed, "Performance Estimation of Distributed Real-time Embedded Systems by Discrete Event Simulations," in *Proceedings of EMSOFT*, 2007.

[4] ——, "A Conservative Approximation Method for the Verification of Preemptive Scheduling using Timed Automata," in *Proceedings of the 15th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2009, pp. 255–264.

[5] G. Madl, S. Abdelwahed, and D. C. Schmidt, "Verifying Distributed Real-time Properties of Embedded Systems via Graph Transformations and Model Checking," *Real-Time Systems*, vol. 33, pp. 77–100, Jul 2006.

[6] G. Madl and S. Abdelwahed, "Model-based Analysis of Distributed Real-time Embedded System Composition," in *Proceedings of EMSOFT*, 2005.

[7] G. Madl, S. Abdelwahed, and G. Karsai, "Automatic Verification of Component-Based Real-Time CORBA Applications," in *Proceedings of the 25th IEEE International Real-Time Systems Symposium (RTSS)*, 2004, pp. 231–240.

[8] S. Pasricha, "Transaction Level Modeling of SoC with SystemC 2.0," in *Synopsys User Group Conference (SNUG)*, May 2002.

[9] A. Ledeczi, A. Bakay, M. Maroti, P. Volgyesi, G. Nordstrom, and J. Sprinkle, "Composing Domain-Specific Design Environments," *IEEE Computer*, pp. 44–51, Nov 2001.

[10] K. Balasubramanian, A. Gokhale, G. Karsai, J. Sztipanovits, and S. Neema, "Developing Applications using Model-driven Design Environments," *IEEE Computer*, vol. 39, pp. 33–40, 2006.

[11] A. Cimatti and E. Clarke and E. Giunchiglia and F. Giunchiglia and M. Pistore and M. Roveri and R. Sebastiani and A. Tacchella, "NuSMV 2: An OpenSource Tool for Symbolic Model Checking," in *Proceedings of the 14th International Conference on Computer-Aided Verification (CAV)*, 2002.

[12] P. Pettersson and K. G. Larsen., "UPPAAL2k," *Bulletin of the European Association for Theoretical Computer Science*, vol. 70, pp. 40–44, feb 2000.

[13] M. Bozga, S. Graf, I. Ober, I. Ober, and J. Sifakis, "The IF Toolset," *Formal Methods for the Design of Real-Time Systems, LNCS 3185*, pp. 237–267, 2004.

[14] ARM, "AMBA Specification rev 2.0, IHI-0011A," 1999.

[15] J. R. W. Muller and W. Rosenstiel, *SystemC Methodologies and Applications*. Kluwer Academic Publishers, 2003.

[16] S. Pasricha, N. Dutt, and M. Ben-Romdhane, "Fast Exploration of Bus-based Communication Architectures at the CCATB Abstraction," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 7, no. 2, pp. 1–32, 2008.

[17] ——, "BMSYN: Bus Matrix Communication Architecture Synthesis for MPSoC," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, (TCAD)*, vol. 26, no. 8, pp. 1454–1464, 2007.

[18] E. Clarke and E. Emerson, "Design and synthesis of synchronisation skeletons using branching time temporal logic," *Logic of Programs, Lecture Notes in Computer Science*, vol. 131, pp. 52–71, 1981.

[19] E. A. Lee, C. Hylands, J. Janneck, J. D. II, J. Liu, X. Liu, S. Neuendorffer, S. S. M. Stewart, K. Vissers, and P. Whitaker, "Overview of the Ptolemy Project," EECS Department, University of California, Berkeley, Tech. Rep. UCB/ERL M01/11, 2001.

[20] Alberto Sangiovanni-Vincentelli, "Defining Platform-based Design," *EEDesign of EETimes*, February 2002.

[21] A. Bonivento, C. Fischione, L. Necchi, F. Pianegiani, and A. Sangiovanni-Vincentelli, "System Level Design for Clustered Wireless Sensor Networks," *IEEE Transactions on Industrial Informatics*, vol. 3, no. 3, pp. 202–214, August 2007.

[22] A. Roychoudhury, T. Mitra, and S. R. Karri, "Using Formal Techniques to Debug the AMBA System-on-Chip Bus Protocol," in *Design, Automation and Test in Europe (DATE)*, 2003, pp. 828–833.

[23] K. W. Susanto and T. F. Melham, "An AMBA-ARM7 Formal Verification Platform," in *International Conference of Formal Engineering Methods (ICFEM)*, 2003, pp. 48–67.

[24] H. Amjad, "Verification of AMBA Using a Combination of Model Checking and Theorem Proving," *Electronic Notes in Theoretical Computer Science, Proceedings of the 5th International Workshop on Automated Verification of Critical Systems (AVoCS 2005)*, vol. 145, pp. 45–61, 2006.

[25] Rafik Henia and Arne Hamann and Marek Jersak and Razvan Racu and Kai Richter and Rolf Ernst, "System Level Performance Analysis - the SymTA/S Approach," *IEE Proceedings on Computers and Digital Techniques*, vol. 152, pp. 148–166, 2005.

[26] K. Richter, M. Jersak, and R. Ernst, "A Formal Approach to MpSoC Performance Verification," *IEEE Computer*, vol. 36, pp. 60–67, April 2003.

[27] Ernesto Wandeler and Lothar Thiele and Marcel Verhoef and Paul Lieverse, "System architecture evaluation using modular performance analysis - a case study," *Software Tools for Technology Transfer (STTT)*, vol. 8, no. 6, pp. 649–667, Oct. 2006.

[28] K. Lahiri, A. Raghunathan, and S. Dey, "System-Level Performance Analysis for Designing On-Chip Communication Architectures," *IEEE Transactions on Computer Aided-Design of Integrated Circuits and Systems*, vol. 20, pp. 768–783, 2001.

[29] C. Ericsson, A. Wall, and W. Yi, "Timed Automata as Task Models for Event-Driven Systems," in *Proceedings of Real-Time Computing Systems and Applications (RTSCA)*, 1999, pp. 182–189.

[30] Venkita Subramonian and Christopher Gill and César Sánchez and Henny B. Sipma, "Reusable Models for Timing and Liveness Analysis of Middleware for Distributed Real-Time Embedded Systems," in *Proceedings of EMSOFT*, 2006, pp. 252–261.

[31] J. Hatcliff, X. Deng, M. B. Dwyer, G. Jung, and V. P. Ranganath, "Cadena: An Integrated Development, Analysis, and Verification Environment for Component-based Systems," in *Proceedings of International Conference on Software Engineering*, 2003.

Gabor Madl received his Ph.D. degree in Computer Science from the University of California, Irvine in 2009. His research is focused on the combination of formal methods and simulations for the model-based analysis of distributed real-time embedded systems. His research was recognized by the 2007 Frank Anger Memorial Award, which he received for promoting the crossover of ideas between the embedded software and software engineering communities. Gabor has worked as an engineer for Google, Fujitsu Laboratories of America, ARM, and has created the open-source DREAM (http://dre.sourceforge.net) and ALDERIS (http://alderis.ics.uci.edu) projects.

Sudeep Pasricha (M'02) received the B.E. degree in electronics and communication engineering from Delhi Institute of Technology, Delhi, India, in 2000, and the M.S. and Ph.D. degrees in computer science from the University of California, Irvine, in 2005 and 2008, respectively.

He has several years of work experience in semiconductor companies, including STMicroelectronics and Conexant. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, Colorado State University, Fort Collins. He has published more than 40 research articles in peer-reviewed conferences and journals and presented several tutorials in the area of on-chip communication architecture design at leading conferences. He recently coauthored a book titled On-chip Communication Architectures (Morgan Kauffman/Elsevier 2008). His research interests are in the areas of multiprocessor embedded system design, networks-on-chip and emerging interconnect technologies, system-level modeling languages and design methodologies, and computer architecture. Dr. Pasricha has received a Best Paper Award at ASPDAC 2006, a Best Paper Award nomination at DAC 2005, and several fellowships and awards for excellence in research.

Nikil D. Dutt (F) received a Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign in 1989, and is currently a Chancellor's Professor at the University of California, Irvine, with academic appointments in the CS and EECS departments. His research interests are in embedded systems, electronic design automation, computer architecture, optimizing compilers, system specification techniques, distributed embedded systems, and formal methods. He received best paper awards at CHDL89, CHDL91, VLSIDesign2003, CODES+ISSS 2003, CNCC 2006, ASPDAC-2006, and IJCNN 2009. Professor Dutt currently serves as Associate Editor of ACM Transactions on Embedded Computer Systems (TECS), and of IEEE Transactions on VLSI Systems (IEEE TVLSI). He served as Editor-in-Chief of ACM Transactions on Design Automation of Electronic Systems (TODAES) between 2004-2008. He was an ACM SIGDA Distinguished Lecturer during 2001-2002, and an IEEE Computer Society Distinguished Visitor for 2003-2005. He has served on the steering, organizing, and program committees of several premier CAD and Embedded System Design conferences and workshops, including ASPDAC, DATE, ESWEEK, CASES, CODES+ISSS, ICCAD, ISLPED and LCTES. He serves or has served on the advisory boards of ACM SIGBED and ACM SIGDA, the ACM Publications Board, and previously served as Vice-Chair of ACM SIGDA and of IFIP WG 10.5. Professor Dutt is a Fellow of the IEEE, an ACM Distinguished Scientist, and an IFIP Silver Core Awardee.

Sherif Abdelwahed is an Assistant Professor with Electrical and Computer Engineering Department at Mississippi State University. He received his Ph.D. degree in Electrical and Computer Engineering from the University of Toronto, Canada, in 2002. From 2000 to 2001, he was a research scientist at Rockwell Scientific Company. From 2002 to 2007 he worked as a research assistant professor with the Institute for Software Integrated Systems at Vanderbilt University. His main research interests include model-based design and analysis of self-managing computation systems, modeling and analysis of distributed real-time systems, automated verification, fault diagnosis techniques, and model-integrated computing. He has published over 70 publications. He is a senior member of the IEEE and member of Sigma Xi.