Formal Performance Evaluation of AMBA-based System-on-Chip Designs

Gabor Madl[1], Sudeep Pasricha[1], Qiang Zhu[2], Luis Angel D. Bathen[1], Nikil Dutt[1]
[1] Center for Embedded Computer Systems
University of California, Irvine, CA 92697
[2] Fujitsu Laboratiories Limited
Kawasaki 211-8588, Japan
[1]{gabe, sudeep, lbathen, dutt}@ics.uci.edu, [2]shu.kyou@jp.fujitsu.com

ABSTRACT

The ARM Advanced Microcontroller Bus Architecture (AMBA) is a widely used interconnection standard for SoC design. In order to support high-speed pipelined data transfers, AMBA supports a rich set of bus signals, making the analysis of AMBA-based embedded systems a challenging proposition. This paper makes two main contributions to the analysis and evaluation of AMBA-based SoC designs. The first contribution is to provide a method for the performance analysis and evaluation of AMBA-based SoC designs using formal models. This method provides a way to obtain the end-to-end execution bounds of AMBA-based SoC designs, and guarantees the correctness of the results. The second contribution is to use these formal models to prove the functional correctness of the SoC designs. Using our formal models, we were able to uncover an ambiguous case in the AMBA specification that can lead to deadlocks. This case has not been previously documented by methods focused on AMBA protocol verification. Finally, we validate the proposed performance analysis approach by comparing results with a SystemC implementation of a digital camera case study.

Categories and Subject Descriptors

I.6.4 [**Simulation and Modeling**]: Model Validation and Analysis; C.4 [**Computer Systems Organization**]: Performance of Systems – *Modeling techniques*

General Terms

Design, Performance, Verification

Keywords

System-on-Chip, Model Checking, Performance Evaluation

1. INTRODUCTION

System-on-Chips (SoCs) are deeply embedded electronic systems operating in resource-constrained environments. It is a well-known fact that the complexity and functionality of these systems often rivals that of high-performance processors from a decade ago, at a fraction of price and energy costs. Bus interconnect standards such as AMBA [3] and CoreConnect [13] are commonly used to integrate heterogeneous components into SoC designs. These bus protocols provide reliable communication in SoC systems by specifying standard methods for interaction between components connected to the bus. Key issues that bus protocols must address include synchronization, dealing with concurrent requests, handling transmission errors, preventing deadlocks, and providing QoS support for SoC designs.

The increasing demand to provide more and more functionality in faster embedded devices requires bus protocols to implement complex methods for component interaction, in order to deal with high-level constraints commonly found in most SoC systems. Modern bus interconnects have to provide support for pipelined data transfers, out-of-order completion, and variable size data bursts. To enforce an increasing number of (often contradictory) application-specific constraints, several restrictions have to be considered by bus protocol designers. On the other hand, bus protocols have to be general enough to be applied in a wide range of applications. As complexity in bus protocols increases, there is a trend to propagate the management of an increasing number of constraints to SoC system designers who implement embedded systems using the protocols. For example, some protocols are known to deadlock or behave unreliably if certain conditions are not met by the SoC system built on a bus-based communication backbone.

Despite the fact that bus protocols have a critical role in providing a reliable platform in SoC systems, their specifications are typically written as a combination of natural languages and timing diagrams. Although this approach is effective in explaining basic use cases to developers, it cannot cover every possible use case, and introduces *ambiguity* in the specification. This ambiguity is especially troublesome when heterogeneous IP blocks have to be integrated on the bus, as different vendors might implement the ambiguous parts of the specification differently. Thus the *interoperability* of such components can be at risk.

Although most vendors provide test vectors to validate and certify whether components work with a bus protocol, there is no well-defined methodology to check whether the system as a whole satisfies high-level design constraints. SoC designers not only have to validate the behavior of components, but they also have to carry the burden of validating their interactions at the transaction-level.

Simulation-based approaches have been found useful in designing large-scale embedded systems. Notable examples include the *OSCI SystemC* [20] standard, that provides an environment for the transaction-level [21, 22] simulation of event-driven SoC systems, and the *SPIRIT Consortium*[1], which aims at creating a standard format for IP exchange among various simulation tools. Simulation-based testing, however, cannot cover every possible use case in SoC systems; it can show the presence of errors, not their absence. Therefore it cannot guarantee the correctness of the results. Moreover, transaction-level simulation is a time-consuming process limiting designers to a few test cases.

This paper proposes a formal method for the performance evaluation of SoC systems built on the ARM Advanced Microcontroller Bus Architecture (AMBA) Advanced High-speed Bus (AHB) [3]. AMBA AHB is a highly successful bus protocol widely used in embedded SoC systems worldwide. The final AMBA 2.0 specification is a 230 page document freely available for download at ARM's website[2]. The AMBA specification is a combination of natural languages and timing diagrams. The specification also describes several use cases that may result in bus deadlocks.

We propose a model checking approach to evaluate the performance of SoC systems based on the AMBA AHB bus. The method is based on the finite state machine model of computation and provides the means for an exhaustive verification of bus interactions. We build on simulation results to obtain parameters for the formal models that we use to explore design spaces that are infeasibly large for transaction-level simulations. We demonstrate our proposed method on a digital camera case study to evaluate the functionality and performance of the system. We use the open-source *NuSMV* [1] model checker for the formal performance analysis. We compare the results with the SystemC simulation of the digital camera case study, at the transaction-level. Our results show that the formal verification complements the simulation results providing a systematic method for the transaction-level validation of SoC designs.

The organization of the paper is the following: Section 2 compares the proposed approach with related work, Section 3 gives a high-level overview of the proposed analysis framework, Section 4 describes the formal modeling of the AMBA AHB protocol, Section 5 describes how we applied this formalism for the functional verification of the digital camera case study, Section 6 demonstrates the formal performance evaluation of the digital camera case study and compares the results of the formal verification with SystemC simulations, and Section 7 presents concluding remarks.

2. RELATED WORK

Simulations are the preferred and widely accepted way to evaluate SoC designs in the industry today. Register-transfer level (RTL) languages such as *VHDL* [14] and *Verilog* [15] are classic hardware description languages widely used in the EDA industry for embedded system design. RTL languages target hardware specification at low-level abstractions providing a synthesizable platform for hardware development. RTL models are usually cycle-accurate and provide high precision for simulations. The low-level abstraction, however, results in slow simulation speeds unsuitable for the analysis of complex SoC systems.

Due to the increase in System-on-Chip design complexity as well as the decrease in the time to market window, today's designers are turning to *transaction-level* modeling languages such as *SystemC* [20], *SystemVerilog* [16], and *SpecC* [9] to perform early design exploration and hardware-software co-design in order to shorten the design cycle. Transaction-level modeling focuses on the *interactions* between systems components, such as bus transfers, interrupts or signals, rather than on gates or registers. Transaction-level languages employ higher-level abstractions than RTL languages and are often not synthesizable.

In this paper we describe a case study that has been implemented using SystemC. SystemC is a library implemented in the C++ language and is a set of features useful for hardware modeling, such as threads, ports, channels, modules, events, processes, etc. SystemC allows the use of cycle accurate, cycle approximate and mixed accuracy modeling abstractions. SystemC employs a logical notion of time and computations are synchronized with respect to a global clock.

A formal approach for evaluating multi-processor system-on-chip (MPSoC) performance has been presented in [23]. The authors extend real-time schedulability methods to analyze generic periodic and sporadic tasks in MPSoC systems at the task-level. In contrast, this paper proposes a more detailed formal analysis method that captures the AMBA AHB bus commonly used in several SoC designs. The architecture of SoCs has a high impact on the overall performance, therefore transaction-level analysis is a necessity for performance evaluation when accurate results are required.

A holistic method is proposed for the schedulability analysis of distributed preemptive real-time systems [27] using a TDMA communication bus. Their work also focuses on the task-level communication and schedulability, therefore it cannot capture the architecture as accurately as the method proposed in this paper.

A generic method for protocol verification using synchronous protocol automata is presented in [7]. The synchronous protocol automata is essentially based on finite state machines. The method proposed in this paper is specific to AMBA-based SoC designs as it formally models the AMBA protocol but could be represented using synchronous protocol automata as well. We have decided to use the NuSMV representation instead as it provides a practical approach for model checking as well.

Other authors have proposed the task-level analysis of scheduling using the timed automata formalism [8]. A generic form to analyze scheduling behavior based on the timed automata model was proposed in [10] for single processor scheduling using the Immediate Ceiling Priority protocol and the EDF algorithm. Although the task-level analysis is more general and could be used for worst-case performance evaluation its results would not nearly be as accurate as the method proposed in this paper that inherently captures the bus communication at the transaction-level.

A method on the functional verification of the PCI protocol is described in [5] using the *Cadence SMV* [18] tool. A similar approach is used to verify the IBM CoreConnect

[1]`http://www.spiritconsortium.com`
[2]`http://www.arm.com`

arbiter in [11]. An early work on applying model checking methods to the AMBA protocol was presented in [24], where the authors used finite state machine models and the SMV tool to uncover an unspecified condition in the AMBA specification. Although the described case study deadlocks the bus, it is due to the flawed implementation rather than the protocol itself. A verification platform for AMBA-ARM7 is presented in [25]. The authors use the SMV tool to prove the functional correctness of the AMBA protocol by checking various properties. The authors do not describe any bugs, rather they focus on properties that have turned out to be valid. A verification platform for AMBA using a combination of model checking and theorem proving is described in [2]. The author extends earlier approaches by considering *both* control and data properties, and describes properties that have proven to be true.

Although previous work has addressed the functional verification of the AMBA protocol, our paper builds on the previous work to propose a systematic formal method for the performance evaluation of AMBA-based SoC designs. Moreover, we have encountered an ambiguous case in the specification that might lead to flawed implementations. Our results do not imply that the AMBA protocol is incorrect, and neither does it imply that the works described in [25, 2] are invalid. Rather, it highlights that even a well-studied and widely used protocol as AMBA AHB does not guarantee that various AMBA-compliant implementations would work interchangeably without any glitches in every case. The main reason for this is the ambiguity of natural languages that should be resolved by future designers by providing a formal specification for their protocol.

3. MODEL-BASED PERFORMANCE EVALUATION USING MODEL CHECKING

We propose a model-based analysis for the performance evaluation of SoC designs. Figure 1 shows a high-level view of the proposed design flow. The design flow starts with the domain-specific model (DSM) that is a high-level specification which specifies key properties of the design, such as its structure, behavior, environment, and key constraints that it has to satisfy. The domain-specific model can be expressed in several ways, using textual specification, timing diagrams, meta-modeling, or other visual methods.

The DSM is then mapped into a formal analysis model. This step is required for any formal analysis unless the DSM is already specified using a formal language (e.g. finite state machines or Petri-nets). The translation requires the necessary details to be abstracted out from the DSM; the formal models usually capture key properties of the system at higher-level abstractions than the DSM itself. As with simulations, the abstraction influences the complexity of the analysis as well as its precision. If the analysis model is too abstract results may become inaccurate, if it is too complex the analysis will likely hit the state space explosion problem. Finding the right abstraction is the key for the successful model-based analysis of SoC systems.

Despite the successful application of formal methods in SoC designs for functional verification, the performance of embedded systems is dominantly evaluated using simulations. Simulations inherently model SoCs with greater accuracy than formal models tailored for verification. Formal verification is essentially an exhaustive state space search

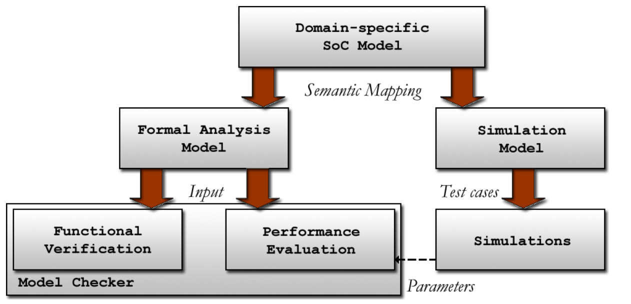

Figure 1: Performance Evaluation of SoCs using Model Checking

therefore it cannot capture the same complexity as a single execution trace using simulations. For complex SoC systems the task-level abstraction is often too coarse, formal models often do not take the system architecture into consideration therefore their applicability is limited for architecture exploration. This paper describes a method that captures SoCs at the transaction-level, thereby creating an abstraction of SoCs that provides a practical balance between analysis accuracy and scalability. The formal models can be used both for functional verification as well as performance evaluation.

In our proposed method, simulation and formal verification are *complementary* techniques for performance analysis, as they address each others weaknesses. Figure 1 illustrates how the combination of simulations and formal verification can be used for the evaluation of SoCs. Designers should perform several tests to obtain results for cases that are representative to the use of the system, and measure execution times of each master, slave, as well as the size of transactions transmitted on the bus. Using this information a formal analysis model can be created that captures key parameters of the system.

We use the simulation and profiling information from the simulations to express the execution times for each master as intervals; the *best case execution time* (`BCET`) refers to the smallest execution time, the *worst case execution time* (`WCET`) refers to the largest execution time found during the simulation and profiling phase. These execution parameters are obtained for each component independently, and they do not include the time spent waiting for the bus or time spent during communication, rather they refer to the computation time of each component. The size of each transaction corresponds to the size of messages written to and read from the slave. Again, we build on the simulation results to approximate the expected size of transactions.

Model checking provides a way to *generalize* the simulation results to predict the worst case behavior of the system. Simulations are a subset of verification: if the system under analysis is completely deterministic a simple simulation can provide guarantees that a system works. Real-life SoCs, however, are usually highly non-deterministic therefore they cannot be efficiently verified by simulations. Formal verification is a technique that allows us to run huge numbers of test cases on the system in short time and prove certain constraints in the system. Simulations cannot provide guarantees on the worst case behavior of a given system as they try to generalize the behavior of the system from a few observed test cases.

We feed parameters to the formal models by creating a model for each master and slave connected to the AMBA bus, and specify their BCET, WCET, the size of the input data read by the master as well as the size of the output data. We might also need to create the necessary synchronization algorithm between the components that controls interrupts and signals in the system. The formal analysis model captures the bus protocol at the transaction-level cycle-accurately. This model provides high accuracy on the bus-level, while building on simulations and profiling to obtain the cycle-approximate behavior of components connected to the bus. We then build on model checking to exhaustively explore all the possible bus accesses that are predictable for the SoC design. This method allows to obtain worst case end-to-end deadlines with high accuracy and formal guarantees by proving that the end-to-end computation time of the system is below a predefined constant bound.

The rest of the paper describes a detailed digital camera case study based on the AMBA AHB bus to demonstrate the proposed design flow shown in Figure 1. Section 4 gives a high-level description of how we have modeled the AMBA protocol using the finite state machine formal model of computation, giving an example of how a DSM (in this case the SoC system based on the AMBA protocol) can be translated into a formal language. Section 5 shows how we applied this formalism for the functional verification of the digital camera case study and describes a previously undocumented ambiguity in the AMBA specification, demonstrating how the formal models can be used for functional verification. Section 6 demonstrates the formal performance evaluation of the digital camera case study and compares the results of the formal verification with SystemC simulations, and Section 7 presents concluding remarks.

4. FORMAL MODELING OF THE AMBA AHB PROTOCOL

This section formalizes the model for the AMBA AHB protocol. The notion of time used in the protocol specification is discrete (bus cycle), therefore the protocol can be represented as a discrete event system. A discrete event system (DES) is a 5-tuple $G = (Q, \Sigma, \delta, q_0, Q_m)$, where Q is a finite set of states, Σ is a finite alphabet of symbols that we refer to as *event labels*, $\delta : Q \times \Sigma \to Q$ is the transition function, q_0 is the initial state, and Q_m is the set of *marker states* (exiting states). A transition or *event* in G is a triple (q, σ, q') where $\delta(q, \sigma) = q'$, $q, q' \in Q$ are the *exit* and *entrance* states, respectively, and $\sigma \in \Sigma$ is the event label. The *event set* of G is the set of all such triples. Events in the DES are untimed and transitions depend only on the current state and the event label.

There are several models of computation that can express discrete event systems, well-known examples include finite state machines, Petri-nets, and data-flow networks. Timed automata and hybrid automata are extensions to finite state machines in order to express the continuous evaluation of system variables, and are therefore too heavyweight to represent cycle-based bus protocols. The discrete event model is simpler than timed automata or hybrid automata models, and offers a more scalable approach for verification by utilizing *Binary Decision Diagrams* [4]. We propose the use of the finite state machine (FSM) model of computation for the representation of the AMBA AHB bus protocol as a discrete event system. We chose finite state machines mainly because they are supported by several model checkers [1, 18, 12].

We have created a cycle-accurate model of the AMBA AHB bus in order to model bus transactions accurately. To ensure that a single split transfer or RETRY response does not deadlock the bus as described in [24], we assume that the arbiter only grants access to a new master when the HRESP signal is OK and the HTRANS signal is IDLE. This introduces an extra cycle arbitration delay in the model. We model arbitration delays, pipelining, and busy slaves (HREADY is 0) in the bus as well. We have also modeled the two-cycle response times for RETRY and SPLIT responses according to the AMBA specification.

```
MODULE master_read_write (BUSREQ, HGRANT, MASTER_STATE,
MASK_MASTER, BCET, WCET, READSIZE, WRITESIZE, START, FINISH)
  VAR
   state :  idle, busreq, haddr, read, write, busy, error;
   prev_state :  idle, busreq, haddr, read, write, busy, error;
   io :  read, write;
   ET : 0..MAXET;
   SIZE : 1..MAXSIZE;
   HADDR : boolean;
   HTRANS : IDLE, NONSEQ, SEQ, BUSY;
   HWDATA : boolean;
  ASSIGN
   init (state) := idle;
   init (io) := read;
   init (SIZE) := 1;
   init (prev_state) := idle;
   next (prev_state) := idle;
  next (state) :=
   case
    HRESP = ERROR : error;
    MASK_MASTER & HGRANT : error;
       HRESP = SPLIT & HGRANT : state;
       !HREADY : state;
       MASK_MASTER : state;
       HRESP = RETRY & HGRANT : prev_state;
       state = idle & START & READSIZE = 0 :  busy;
       state = idle & START : busreq;
       state = idle :  idle;
       state = busreq & HGRANT : haddr;
       state = busreq & !HGRANT : state;
       state = haddr & HGRANT : read;
       state = read & HGRANT : write;
       state = write & HGRANT & SIZE = READSIZE & io = read :
busy;
       state = write & HGRANT & SIZE = WRITESIZE & io = write :
idle;
       state = write & HGRANT & SIZE < READSIZE & io = read :
haddr;
       state = write & HGRANT & SIZE < WRITESIZE & io = write :
haddr;
       state = busy & ET < BCET : busy;
       state = busy & ET = WCET : busreq;
       state = busy & BCET <= ET : busy, busreq;
       1:  error;
     esac;
...
```

Figure 2: Partial NuSMV Finite State Machine Model for an AMBA AHB Master

```
MODULE slave (HADDR, HTRANS, HWDATA, HRDATA, HREADY, HRESP,
HMASTER, HSPLIT, MASK_MASTER1, MASK_MASTER2, MASK_MASTER3,
SLAVE_STATE)
   VAR
     state :  {idle, write, read, error};
     prev_state :  {idle, write, read, error};
     extended :  boolean;
   ASSIGN
     init (state) := idle;
     init (prev_state) := state;
     init (extended) := 0;
     next (prev_state) := state;
     next (state) :=
       case
         SLAVE_STATE != x :  SLAVE_STATE;
         HRESP = SPLIT : idle;
         !HREADY : state;
         HTRANS = BUSY : state;
         HRESP = RETRY : prev_state;
         state = idle & HTRANS = NONSEQ & HADDR : write;
         state = idle :  state;
         state = write & HTRANS = NONSEQ : read;
         state = read & HTRANS = NONSEQ & HWDATA : idle;
         1 :  error;
       esac;
...
```

Figure 3: Partial NuSMV Finite State Machine Model for an AMBA AHB Slave

We have chosen to implement a round-robin arbiter, mainly to avoid starvations that might arise when a fixed-priority arbiter is used. We consider `RETRY` responses from the slave (`HRESP = RETRY`), as well as split transfers. We do not, however, model locks on the transfer (`HLOCKx` signals).

Figure 2 shows how we modeled a generic AMBA AHB master as a finite state machine. We have used the NuSMV syntax for its compactness. We identify six states (`idle, busreq, haddr, read, write, error`) as shown in Figure 2. Transitions are specified within the `case ...esac;` block. Transitions are ordered deterministically; the next value of `state` will be specified by the first guard that evaluates to `true`. The figure is only partial; the `HADDR`, `HTRANS`, `HRDATA`, `HWDATA`, and `BUSREQ` signals depend on the `state` of the master. The `MASTER_STATE` variable is used for the SoC evaluation described in Section 6 to provide a fast and simple method to track the master's current state from the arbiter. The `BCET` and `WCET` parameters are given as inputs to the AMBA AHB master. The `READSIZE` and `WRITESIZE` parameters specify the size of the data read from and written to the bus. The `BCET`, `WCET`, `READSIZE`, and `WRITESIZE` parameters are provided by the simulations.

Figure 3 shows how we modeled a generic AMBA AHB slave using NuSMV. We have identified four states (`idle, write, read, error`) as shown in Figure 3. Transitions are specified within the case block. The transitions of the slave have to be synchronized with the master – *e.g.* the slave has to be in the `read` state when the master is in the `write` state – otherwise the slave (and the master) will enter the `error` state. Figure 3 is only partial; the `HREADY` and `HRESP` signals are assigned values non-deterministically for the functional verification. The slave records split transactions by storing the master's address in the `MASK_MASTER1`, `MASK_MASTER2`, and `MASK_MASTER3` flags. These flags are managed by the slave (the arbiter also maintains its own flags for which master is masked) and are cleared when the slave issues an `HSPLITx` signal. The `extended` variable is used to extend the duration of `RETRY` and `SPLIT` responses for two clock cycles according to the AMBA specification.

We have modeled a round-robin arbiter to evaluate the case studies described in Section 5 and Section 6. We have not included a detailed description of the arbiter design, mainly because (1) it is not generic, rather a specific implementation that corresponds to the AMBA specification, (2) the design is too complex to fit within this paper. For the NuSMV models used in the formal analysis please visit `http://alderis.uci.edu/amba2`.

5. FUNCTIONAL VERIFICATION OF AMBA-BASED SOC DESIGNS

Although our major goal is to evaluate the performance of SoC designs using the formal FSM models these models also allow for the functional verification of AMBA AHB-based SoCs. Despite the fact that the functional verification of the AMBA AHB protocol has been addressed before by various researchers [24,25,2], we were able to uncover an ambiguity in the protocol that has not yet been documented. This section describes how we used the NuSMV tool to verify deadlock-freedom and liveness properties in a single bus system with three masters and one slave, using round-robin arbitration. In the discrete event model a *deadlock* can be observed as a state with no transitions enabled. A *livelock*, on the other hand, refers to a state from which only a subset of all the states is reachable, that cannot provide the necessary functionality for the system.

The AMBA protocol allows three types of responses by the slave: `OK` signals that the transaction in the previous clock cycle has been successfully completed, `RETRY` signals that the slave wants the master to repeat the transaction from the previous clock cycle, and `SPLIT` is a signal to the arbiter to mask the master. A slave issues the `SPLIT` response when it predicts that it will be unable to receive data – a rather ambiguous definition in the AMBA specification. Later, a slave can signal the arbiter using the `HSPLITx` signal that it is now ready to process data and requests that the arbiter unmasks a previously masked master.

Figure 4 shows the design of the case study. We have used the open-source NuSMV model checker to verify CTL [6] properties on the finite state machines. During this process we have discovered several trivial deadlock cases that are covered by the AMBA specification. For example, we were able to show that a `SPLIT` response followed immediately by a `RETRY` response deadlocks the system, as the master receiving the `RETRY` response has not started transmitting on the bus yet. The AMBA specification, however, requires the slave to issue an `OK` response following the `SPLIT` response. Similarly, we found that the combination of a `RETRY` response and a low `HREADY` signal may deadlock the bus because the master is required to keep its state when the `HREADY` signal is low, but is also required to repeat the last transmission since the response is `RETRY`. These ambiguities, however, do not have a high practical value as the combination of these signals does not seem to be logical in a real-life SoCs.

We were able, however, to uncover a non-trivial ambiguity that might lead to flawed implementations. Consider an

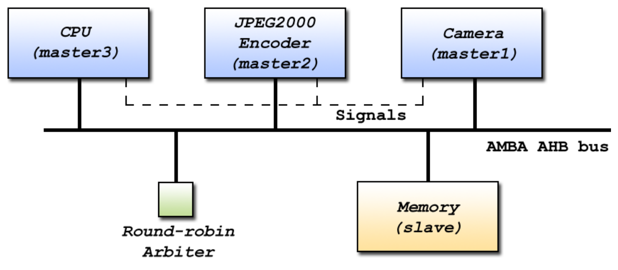

Figure 4: Digital Camera Case Study based on the AMBA AHB Bus

SoC system based on the AMBA AHB protocol, using two masters (`master_1, master_2`) and a slave. The arbiter has to keep track of the masters' state in order manage the split transfers. This could be implemented by providing dedicated wires between the masters and the arbiter, however this is impractical in most cases as it requires extra computation and hardware. An alternative method is to monitor the bus traffic to obtain the master and slave states. The arbiter may use the `HTRANS` signal to check whether the master is idle or transmitting (`NONSEQ, SEQ`), the `HBURST` signal to predict the remaining cycles from the transfer, and the `HRESP` signal to monitor whether the active master and slave has to step back to repeat a transaction.

Let's assume that the slave have previously split `master_1` (`master_1` is masked by the arbiter), and is in a transaction with `master_2`. The slave can unmask a master by issuing an `HSPLITx` signal using the masked master's address to the arbiter. Consider that the slave tries to unmask master_1 by setting `HSPLITx` when it issues a `RETRY` response. The AMBA specification is ambiguous on what the arbiter should do in this case. The specification says that a master has to repeat the last transaction when it receives the `RETRY` response. If the arbiter monitors the bus signals to keep track of the masters' states it will try to go back to its previous state to keep synchronized with the master and the slave. However, if the arbiter implements this behavior it will not unmask `master_1` as the client requests. Since there is no acknowledgement for `HSPLITx` signals the client thinks that `master_1` is already unmasked, and won't request that the arbiter unmasks it again. This may result in deadlock as `master_1` never gets access to the bus again. The AMBA specification states, that "A slave which issues RETRY responses must only be accessed by one master at a time." The authors of this paper could not reach an agreement whether the "access" refers to access through the bus or access by being split by the slave - which would cover this deadlock, but would also imply that a slave cannot issue a `RETRY` response if it may split a master.

Figure 4 shows the architecture of the digital camera case study used throughout the paper. The digital camera consists of three masters (`CPU, JPEG2000 Encoder, Camera`), and one slave (`Memory`). We have used a round-robin arbiter for this case study. A default master gets access to the bus when none of the three regular masters request the bus. Section 6 describes the behavior of the system in more detail. For the functional verification we have considered the case when all the masters are allowed to concurrently request access to the bus and carry out read/write transactions in an arbitrary unsynchronized manner. The proposed functional verification does not take the internal computation of components into consideration, rather it treats them as "black boxes" that use the bus according to the specification. The results described in this section are therefore applicable to any SoC that uses the architecture show in Figure 4 with a round-robin scheduler.

Using the NuSMV model checker we were able to prove several properties in the digital camera case study shown in Figure 4. First we showed that the `error` state is unreachable in all the masters and the slave by using the CTL formulas (x refers to the index used for all masters):

```
AG (masterx.state != error),
AG (slave.state != error).
```

The AMBA protocol permits a simple way for livelock by allowing the slave to arbitrarily split masters. If the slave splits two masters and does not unsplit them, we end up in a livelock condition as the split masters never get a chance to serve requests. Moreover, if the slave splits all the masters and does not unsplit them the system deadlocks. We showed these conditions by checking the following CTL formulas:

```
EF (MASK_MASTERx & MASK_MASTERy),
EF (MASK_MASTER1 & MASK_MASTER2 & MASK_MASTER3).
```

We had to specify rules within the slave that enforce that whenever two masters are split one of them will eventually be unsplit. We have verified this property using the following CTL formulas:

```
AG ((MASK_MASTERx & MASK_MASTERy) ->
AF (!MASK_MASTERx | !MASK_MASTERy)),
AG ((MASK_MASTER1 & MASK_MASTER2 & MASK_MASTER3) ->
AF (!MASK_MASTER1 | !MASK_MASTER2 | !MASK_MASTER3)).
```

We have decided to use the round-robin arbiter to ensure that no starvations occur within the system. We were able to show that all bus requests by the masters eventually get served by the arbiter by checking the following CTL formulas:

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

These formulas ensure that the system does not deadlock or livelock and works properly. In order to prove these formulas we have assumed that the following formulas evaluate to true infinitely often (using the `JUSTICE` NuSMV keyword): `HREADY`, `HRESP = OK`, `HSPLIT = master1`, `HSPLIT = master2`, `HSPLIT = master3`. This was necessary to avoid trivial erroneous cases such, as when the slave is never ready to receive data, or when it continuously sends `RETRY` responses. By disallowing the simultaneous use of the `HRESP = RETRY` and the `HSPLITx` signal – that have caused a deadlock as described above – we were able to show that the system works correctly.

6. PERFORMANCE EVALUATION OF AMBA-BASED SOC DESIGNS

This section explains the digital camera case study shown in Figure 4 and Figure 6 in detail, and describes the proposed method for performance evaluation that combines the transaction-level simulation approach with model checking. The digital camera used for the case study implements the new *JPEG2000* [17] still image compression standard developed by the JPEG committee. The advantages of JPEG2000 over its predecessor *JPEG* include lossy to lossless compression, region of interest (ROI), multiple resolution representation, error resiliency, etc. The JPEG2000 encoder is divided into three main parts: image transformation, quantization and entropy coding. Unlike JPEG, which relies on the more commonly used discrete cosine transform (DCT), JPEG2000 uses the *Discrete Wavelet Transform (DWT)*. JPEG2000's choice of entropy coding is based on the *Embedded Block Coding with Optimal Truncation (EBCOT)* [26].

The SystemC model used for this project is part of an ongoing effort to develop a suite of SystemC models. Our JPEG2000 encoder is a transaction-level implementation of the proposed architecture described in [28]. The implemented simulation model is semi-cycle accurate, in the sense that it is cycle accurate at the transaction-level, but the functional part is cycle approximate.

6.1 JPEG2000 Encoder Description

Figure 5 shows the block diagram for the JPEG2000 encoder. Designers have the option of implementing a distributed compression method, where the image is broken up into tiles, and the compression is carried out for each tile separately. Although this is feature is not required by the JPEG2000 specification we have decided to implement it to improve the concurrent processing in the system and thus the overall performance of the SoC. Tile size varies, from smaller sizes – 64x64 pixels for memory restrained designs – to 512x512 for better compression quality. These parameters vary and designers need to consider the requirements for their specific designs.

After the image is tiled, each tile is passed though the *DC Level Shifting* step which converts the tile pixels from unsigned integers to two's complements. In the next step the tile as passed through the *Multi-Component Transform (MCT)*, which is in charge of transforming the input tile from RGB color format to either YUV by using the reversible color transform (RCT), or to YCbCr by using the irreversible color transform (ICT). RCT can be used in both lossless and lossy compression whereas ICT can only be used for lossy compression. After the tile has been transformed, it is processed by the discrete wavelet transform (`DWT`), which further decomposes the tile into different levels of decomposition. For every pass `DWT` makes on a tile, depending on the number of decomposition levels needed, `DWT` generates four sub-bands, denoted as LL, HL, LH, and HH, where LL represents the downsampled tile (half the width/height of the previous tile), and the other three sub-bands represent a residual version of the tile which are used for the image reconstruction process. Once `DWT` has processed the tile it is passed through the quantization step only when lossy compression is used. The user has the option to declare Regions Of Interest (ROI), that are encoded independently from the rest of the image based on user specifications. This allows to use lossless compression for some (interesting) parts of the

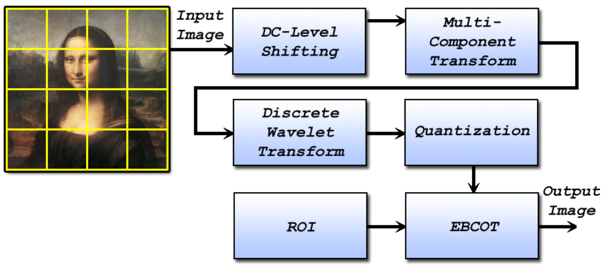

Figure 5: JPEG2000 Encoder Block Diagram

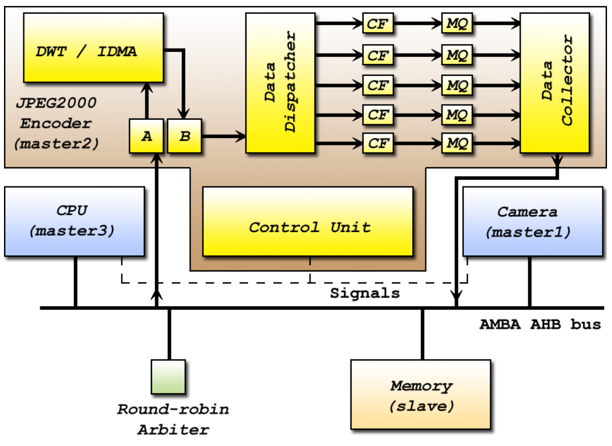

Figure 6: The JPEG2000 Encoder Integrated in the Digital Camera Case Study

image, while using lossy compression for the rest of the image. Finally, the image (or tile) is processed with EBCOT, which produces the final bitstream for the image.

EBCOT can be further subdivided into two parts, commonly known as *Tier-1* and *Tier-2*. Tier-1 is the most computation intensive part of JPEG2000. Because of the complexity of both DWT and Tier-1, most designers choose to implement these functionalities in hardware. Tier-2, however, is very control intensive, therefore it is often implemented in software on the main CPU. The case study shown in Figure 6 is influenced by these observations.

In the proposed architecture shown in Figure 6, the `DWT` module has an internal DMA engine (iDMA) that fetches the tiles from main memory to either `DWT`'s local memory or bank `A` of the tile memory, depending on whether `DWT` is currently processing a tile or not. The `DWT` module is capable of lossless and lossy compression and implements DC Level Shifting and MCT. After `DWT` is done processing the tile, it writes its transformed image to bank `B` in tile memory if and only if `Data Dispatcher` has finished fetching all codeblocks for the previous tile from bank `B`. Therefore, the `DWT` module may be blocked by the slower `Data Dispatcher`.

The `Data Dispatcher` module reads the codeblocks from bank `B` in the tile memory and performs the quantization step on them. Its main job is to feed bitplanes onto each *Bit Plane Coder (BPC)* so that at any given time, there could be up to N different bitplanes being processed by the BPC modules. A BPC is actually the module in charge of performing Tier-1 on incoming DWT coefficients, and it is subdivided into two parts, the *Context Formatter (CF)* and the *Arithmetic Coder (MQ-Coder)*. These blocks are denoted as `CF` and `MQ` in Figure 6. Finally, the processed data is collected by the *Data Collector*, from which it is written to the main memory through the AMBA AHB bus.

6.2 Simulation-based Evaluation

Our simulation abstraction is cycle accurate at the transaction-level, and the functional behavior of each module is "cycle-count-accurate". Each one of the blocks in Figure 6 is implemented as `SC_MODULE` which is a special class in SystemC used to declare modules. Communication between modules is implemented through `SC_PORTS` using `SC_SIGNALS`.

Within each `SC_MODULE` there may be several concurrently executing threads, declared as `SC_THREAD` in SystemC. For instance, `DWT` has a tiling engine thread that emulates iDMA and fetches the tiles from main memory, a compute thread that emulates the `DWT` lifting kernel and wakes up when the controller signals that there is a tile ready to be processed, a read thread that fetches tiles from tile memory, and a write thread that writes `DWT` coefficients to tile memory. The `Data Dispatcher` has two threads, one that reads `DWT` coefficients from tile memory and the main data dispatcher thread that distributes the bitplanes among all of the bit plane coders in round robin fashion.

The `Context Formatter` and the `MQ-Coder` both have three separate threads, a read (from input FIFO) thread, a write (to output FIFO) thread and a compute thread. The data collector module also has two threads, one for reading from the bit plane coder output FIFOs in round robin fashion, and one for writing the encoded data back to main memory. The application consists of 42 threads overall - a rather large number for a simple SoC design. The exhaustive verification of the SystemC model is practically infeasible due to the large degree of non-determinism present in the simulation models. A recent paper summarizes the problems arising from the complexity caused by the inherent non-determinism of multi-threaded embedded systems [19].

The SystemC model is configured using a configuration script that sets up its parameters based on the input image that the model will process. The parameters include tile width, tile height, image width, image height, `DWT` decomposition levels, etc. The script configures and runs the model for a given amount of test images. From each simulation run we obtain the execution intervals for processing tiles, and the size of compressed tiles sent over the AMBA AHB bus.

Table 1 and Table 2 show the parameters that we have obtained by the SystemC simulations. We have run simulations on five different pictures using 128×128 pixel images as input for the compression. The Tier-1 columns describe the measured execution time of the Tier-1 JPEG2000 compression in bus cycles, Tier-2 columns correspond to the software implementation of Tier-2 on the main CPU. The Input column shows the size of the tile as input to the `DWT` and Tier-1, Output specifies the worst case size of the tile after the compression in Tier-1. These parameters are used for the formal evaluation of worst case end-to-end execution times as described in Section 6.3, that is larger than any of the end-to-end times measured using the simulations shown in Figure 1 and Table 2.

6.3 Model Checking-based Performance Evaluation

This section describes how we utilized model checking to evaluate the worst case behavior of the digital camera case study shown in Figure 6 based on the simulation results described in Subsection 6.2 above. The simulations give us very accurate results for the end-to-end processing of image tiles, however they can only cover a few execution traces of the system. The formal model checking approach provides the means to evaluate larger design spaces to obtain the *worst case end-to-end execution time* of the overall SoC design. As discussed in Section 4 the formal models used for the evaluation are cycle-accurate on the bus transaction-level.

We have shown in Section 5 how we proved the overall functionality of the system. The FSM models used for the performance evaluation are more lightweight that the models described in Section 4; we do not consider split transactions, `RETRY` responses, or blocking slaves (`HREADY` is assumed to be high), as these functionalities are not used in the digital camera case study shown in Figure 6. Although these assumptions are not required for the performance analysis they increase the scalability of the model checking.

We have used simple Boolean variables to model the interrupts and signals in the digital camera, thus enforcing the dependencies between components. Although finite state machine is inherently an untimed model of computation it can capture time on a discrete time scale as transitions can be ordered. In our analysis we have declared a global time variable that is increased at every cycle. The performance analysis can then be expressed as a reachability problem: when `Master_3` has written its data to the memory is the execution time always smaller than some value x? Formally using CTL formula: `AG (finish3 -> TIME <` x`)`, where `finish3` is the signal generated by `Master_3` when it is in the `write` state and it has written all the required data to the memory. The NuSMV model checker exhaustively explores all the execution traces that are valid in the FSM model in order to show whether the formula is correct or not.

As seen in Table 1 and Table 2 the execution intervals are in the order of millions of cycles. The execution intervals for system components introduce a large degree of non-determinism in the system; there are around $10^6 \times 10^6 = 10^{12}$ valid execution traces (but several more states) in the system based on the simulation data obtained during the simulation phase. The practical limit on the state space size of analyzable systems using state-of-the-art methods today is in the order of 10^{20} states. These numbers suggest that the exhaustive cycle-accurate model checking of the digital camera SoC design is infeasible as we are likely to face the state space explosion problem. To overcome this problem we have increased the timescale of the simulation (from cycles to 1000 cycles), thereby creating an abstraction of the system that is cycle-approximate with the highest precision available without hitting the state explosion problem. We have assumed that the `Camera` (`Master_1`) requests access to the bus without delay as it does not do any data processing.

We have considered around 100 different execution traces for `Master_2` and around 1000 execution traces for `Master_3` in the SoC design shown in Figure 6, therefore we have evaluated 10^5 execution traces of the digital camera case study.

The proposed method is computationally intensive, and

Table 1: Simulation Results for JPEG2000 Encoding using 64×64 pixel Tiles (for a single tile). Scale: cycles

Image	`DWT ET`	Tier-1 `BCET`	Tier-1 `WCET`	Tier-2 `ET`	Input	Output	End-to-end `WC`
baboon	194 188	517 005	741 519	9 122 240	12 288	11 099	10 335 043
boat	194 188	165 141	737 046	8 750 875	12 288	10 046	10 044 857
goddesses	194 188	513 846	772 461	8 663 630	12 288	11 456	9 996 487
goldhill	194 188	242 055	747 954	8 672 436	12 288	10 376	9 978 464
lena	194 188	461 601	769 239	8 689 815	12 288	11 979	10 024 198

Table 2: Simulation Results for JPEG2000 Encoding using 128×128 pixel Tiles. Scale: cycles

Image	`DWT ET`	Tier-1 `BCET`	Tier-1 `WCET`	Tier-2 `ET`	Input	Output	End-to-end `WC`
baboon	751 393	2 315 254	3 151 948	9 010 373	49 152	36 537	14 290 609
boat	751 393	1 764 568	3 086 892	8 758 372	49 152	41 719	13 990 027
goddesses	751 393	1 843 190	3 219 664	9 451 990	49 152	42 391	14 823 509
goldhill	751 393	2 325 098	3 173 076	8 768 459	49 152	41 645	14 090 307
lena	751 393	2 364 360	3 241 400	8 793 070	49 152	37 578	14 172 351

Table 3: Parameters used for Performance Evaluation by Model Checking. Scale: 10^4 cycles

tile size	`Master_1 ET`	`Master_2 BCET`	`Master_2 WCET`	`Master_3 BCET`	`Master_3 WCET`
64×64	1	35	97	866	913
128×128	1	251	400	875	946

Table 4: Worst Case Bounds on the End-to-end Computation Time of the Digital Camera Case Study obtained using Model Checking. Scale: cycles

Tile size	`WCET`
64×64 pixel tiles	11 000 000
128×128 pixel tiles	17 000 000

its performance degrades exponentially with respect to the state space size of the analyzed system. Evaluating the worst case performance of the digital camera case study shown in Figure 6 using 64x64 tiles and 10^4 bus cycles resolution takes around 6 hours on a dual AMD Opteron 240 (1.4GHz) computer using 1GB main memory using Linux OS, and 4 days using 128x128 tile sizes. Table 4 summarizes the results of formal model checking on the worst case execution bounds of the digital camera case study. To evaluate the accuracy of the formal evaluation we have tried to find a tight lower bound on the worst case execution time. We found that the worst case end-to-end computation of the digital camera SoC using 64x64 tiles is greater than 10 440 000 cycles. This shows that our results are rather tight. Finding the tightest bound using the proposed method is theoretically possible, but in practice it is rather time consuming therefore we have not considered finding the tightest bound an objective for this case study.

Our experiments confirm that the method is computationally intensive. This might present scalability issues when trying to apply the method for large-scale SoCs. In this case, the combination of several techniques may be used. First, we might increase the scalability by increasing the timescale of the analysis, at the loss of some precision. Second, the method can be hierarchically composed. End-to-end execution times for SoCs can also be represented as intervals thus providing a way to encapsulate larger SoC designs as a single component. Third, we can limit how many execution traces we want to capture in the models *e.g.* by using constant execution times. As with any method, there is a tradeoff between analysis accuracy and performance.

Comparing the results of the formal analysis shown in Table 4 with the results of the SystemC simulations shown in Table 1 and Table 2 shows that the results are rather close, but the simulation-based analysis did not find the actual worst case execution bound, so using simulations to estimate the worst case performance of the system could be overly optimistic. Our results show that the formal performance analysis is able to provide tight worst case execution numbers for the end-to-end processing of the digital camera SoC, given the right parameters.

The performance evaluation results shown in Table 4 assume that the digital camera is initially in the idle mode, meaning that all components are ready to serve requests as soon as they receive them. In order to show that the performance of the system does not degrade during the compression we have modeled the processing of four subsequent 64x64 pixel tiles in the system. We were able to show that the overall computation time is within the $4 \times \texttt{WCET} = 44\ 000\ 000$ cycle bound. In the actual implementation the computation of the `DWT`, and the `CF`, `MQ` components may overlap, and Tier-1 is essentially a pipeline architecture therefore we would rather expect performance increase than degradation for the end-to-end processing of multiple tiles as the compression of tiles overlaps. Evaluating the worst case performance of the digital camera case study for four subsequent tiles using 64x64 pixel tiles took us around 36 hours on the AMD Opteron machine mentioned above. We have omitted the analysis of 128x128 tiles for subsequent tiles as the results depend on the system architecture, rather than the tile size.

The proposed formal performance evaluation method is unique compared to simulation-based evaluations as it covers orders of magnitude larger design spaces. The application of the method allows designers to avoid the common mistake of underestimating the worst case performance of SoCs as a result of inadequate coverage by simulations.

7. CONCLUDING REMARKS

We have presented a formal method to evaluate the functionality and performance of SoC designs based on the AMBA AHB bus. We have described a previously undocumented ambiguity in the AMBA specification that might lead to flawed designs. The proposed formal method for performance evaluation combines simulations with model checking to provide a way for high accuracy design space exploration. The formal models inherently capture bus communication at the transaction-level, thereby creating an abstraction with a practical balance between analysis accuracy and scalability. The proposed formal performance analysis can be used to obtain the worst-case bounds on the end-to-end execution time of SoCs and – unlike simulations – guarantees the correctness of the results. Our experiments with a digital camera SoC demonstrate the applicability and high accuracy of the method. The formal evaluation described in this paper could be fully automated using existing model checkers. Comparisons with state-of-the art simulation techniques show that the proposed method can be efficiently used for the systematic transaction-level validation of SoC designs.

8. ACKNOWLEDGEMENTS

We would like to thank Professor Fadi J. Kurdahi for providing the JPEG 2000 architecture design and his help in developing the SystemC simulation models. This research was partially supported by a CPCC Fellowship.

9. REFERENCES

[1] 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)*, 2002.

[2] H. Amjad. Verification of AMBA Using a Combination of Model Checking and Theorem Proving. *Electronic Notes in Theoretical Computer Science*, 145:45 – 61, 2006.

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

[4] R. E. Bryant. Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

[5] P. Chauhan, E. M. Clarke, Y. Lu, and D. Wang. Verifying IP-Core based System-On-Chip Designs. In *IEEE ASIC SOC Conference*, pages 27 – 31, 1999.

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

[7] V. D'silva, S. Ramesh, and A. Sowmya. Synchronous protocol automata: a framework for modelling and verification of SoC communication architectures. In *IEEE Proceedings of Computers and Digital Techniques*, volume 152, pages 20 – 27, January 2005.

[8] C. Ericsson, A. Wall, and W. Yi. Timed Automata as Task Models for Event-Driven Systems. In *Proceedings of RTSCA '99*, 1999.

[9] D. Gajski, J. Zhu, R. Domer, A. Gerstlauer, and S. Zhao. *SpecC: Specification Language and Methodology*. Kluwer Academic Publishers, 2000.

[10] T. Gerdsmeier and R. Cardell-Oliver. Analysis of Scheduling Behaviour using Generic Timed Automata. *Electronic Notes in Theoretical Computer Science*, 42, 2001.

[11] A. Goel and W. R. Lee. Formal Verification of an IBM CoreConnect Processor Local Bus Arbiter Core. In *DAC '00: Proceedings of the 37th conference on Design automation*, pages 196–200. ACM Press, 2000.

[12] G. J. Holzmann. *The SPIN model checker: Primer and reference manual*. Addison Wesley, 2004.

[13] IBM. 32-bit Processor Local Bus Architecture Specifications ver 2.9, SA-14-2531-01, 2001.

[14] IEEE. VHDL (IEEE 1076 Standard), 2000.

[15] IEEE. Verilog (IEEE 1364 Standard), 2001.

[16] IEEE. SystemVerilog (IEEE 1800 Standard), 2005.

[17] JPEG committee. ISO/IEC JTC1/SC29/WG1 N1855, JPEG 2000 Part I: Final Draft International Standard (ISO/IEC FDIS15444-1). 8.2000.

[18] K.L. McMillan. The SMV system. Technical Report CMU-CS-92-131, 1992.

[19] E. A. Lee. The Problem with Threads. *IEEE Computer*, 39(5), May 2006.

[20] OSCI. SystemC ver 2.1 (IEEE 1666 Standard), 2005.

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

[22] S. Pasricha, N. Dutt, and M. Ben-Romdhane. Extending the Transaction Level Modeling Approach for Fast Communication Architecture Exploration. In *Design and Automation Conference (DAC)*, 2004.

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

[24] 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)*, page 10828, 2003.

[25] K. W. Susanto and T. F. Melham. An AMBA-ARM7 Formal Verification Platform. In *ICFEM*, pages 48–67, 2003.

[26] D. Taubman. High performance scalable image compression with EBCOT. *IEEE Transactions on Image Processing*, 9:1158 – 1170, July 2000.

[27] K. Tindell and J. Clark. Holistic Schedulability Analysis for Distributed Hard Real-Time Systems. *Microprocessing and Microprogramming - Euromicro Journal (Special Issue on Parallel Embedded Real-Time Systems)*, 40:117–134, 1994.

[28] C. Zhang, Y. Long, and F. J. Kurdahi. A Scalable Embedded JPEG2000 Architecture. In *Proceedings of the Workshop on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, pages 334–343, 2005.