A Highly Efficient Modeling Style for Heterogeneous Bus Architectures

Manoj Ariyamparambath, Denis Bussaglia, Bernd Reinkemeier
Intellectual Property and Design Services,
Synopsys Inc, Germany

Tim Kogel, Torsten Kempf
Integrated Signal Processing Systems
Aachen University of Technology,

Abstract—**The ever increasing complexity and heterogeneity of modern System-on-Chip designs demands validation of the system performance as early as possible. The on-chip bus architectures play an important role to meet the design performance. Today many heterogeneous on-chip bus architectures are defined to address the design exploration.**

In this paper we introduce an efficient modeling style of heterogeneous bus architectures at high levels of abstraction. We capture different bus architectures by using a generic, parameterizable bus model, which captures performance issues without significant loss of accuracy.

Our modeling style is based on the SystemC language, a special channel library and attached coding style. The combination provides the ground layer for the efficient and fast simulation, which in turn enables the validation of the functionality and performance of the system at high abstraction levels.

The approach has been successfully used from defining the Executable Specifications at the functional level to the architecture explorations with HW/SW integration for an IPv4 Router with Quality of Support, design example.

Index Terms—**Heterogeneous bus architecture, System level design, SoC, SystemC.**

I. Introduction

In the SoC era, with the increased complexity of the systems the process of specification, partitioning and the verification of the system has a decisive impact on the success of the design. Increasing system complexity demands fast validation of the concepts and the possible architectures of the design.

The typical SoC design may contain one or more processors, hardware accelerators for the dedicated functions linked together with complex on-chip bus architectures. The SoC communication architecture determines the way the different functional units are synchronizing and exchanging data and has a great impact on the system's performance [1].

Intellectual Property (IP) re-use of the bus architectures creates a framework for the efficient realization of the complex SoCs. The dominant communication architectures in the industry use shared bus with prioritized arbitration. The main players are ARM AMBA^{TM} and IBM CoreConnect^{TM}.

The corresponding cycle accurate Transaction Level Model (TLM) IP [4, 6] of the respective bus addresses the design exploration and system level modeling for this kind of homogenous communication architectures

Other than this, today heterogeneous and application domain specific bus architectures are defined to meet complex performance requirements. The cycle-level TLM approach fails to support exploration across heterogeneous bus architectures, since these models are bus-specific. Hence the evaluation of the different bus architecture requires tedious re-writing of the interfaces.

Our approach to system level modeling of heterogeneous bus architectures is to move to higher abstraction levels and use a generic, parameterizable bus model for communication architecture exploration. We have conceived a *packet-level TLM* style, which captures different bus architectures through parameterization of a single generic bus template.

The SoC designs for the considered networking application domain [2,3] involve complex algorithms and protocol handling in SW as well. The functional models used in the generic packet-level bus platform are re-used when the SW functionality is integrated with an Instruction Set Simulator (ISS) of the target processor. The integration of the ISS calls for a cycle accurate bus platform. The functional models are interfaced to the cycle accurate TLM of the bus through adapters, which are specifically developed for the intended bus architecture.

II. Related Work

Bus based architectures are very commonly used to facilitate the communication between the various functional blocks of the design [5]. Many studies and efforts are diverted into the communication architectures, as they are the key to high performance, low power systems. The ability to re-use the IP reduces the design risk and the time to market.

Many commercial on-chip bus architectures are available based on the IP re-use [6, 7]. These popular bus architectures are based on the priority arbitration schemes. Along with this commercial IPs there are new variants of application domain specific communication architectures developed by the academic world [8,9].

For the bus based architecture exploration and system level design many commercial [4, 6] and academic solutions [11,12] are available which provides cycle based TLM or component based [10] design methodologies and tool environments.

The unique contribution of this paper is to further raise the abstraction level to a packet-level TLM style and create a corresponding generic bus template, which enables

This work was done in association with Institute for Integrated Signal processing Systems, Aachen University. The authors are with Synopsys Inc. (manu, denisb, reinke)@synopsys.com and Aachen University (kogel, kempf)@iss.rwth-aachen.de

architectural exploration across heterogeneous bus architectures. The case study shows excellent results w.r.t simulation speed, modeling efficiency and accuracy.

III. High Abstraction Modeling – Packet-Level TLM

Predominantly, the approach to the system-level performance analysis is based on simulation of the entire system. The models and the communication architectures can be modeled at different abstraction levels. Generally the use of abstraction is a trade-off between simulation speed and accuracy. In this section we highlight the abstraction levels enabled by SystemC 2.0 and then elaborate on the packet-level TLM.

A. *Transaction Level Modeling (TLM) Overview*

One of the primary goals of SystemC 2.0 [13] is to raise the communication abstraction and enable system level modeling with Transaction Level Modeling [14] style. At this level, the communication is modeled using the function calls, thus hiding the lower level implementation details. The resulting improvements in simulation speed and modeling efficiency enables the system architect to create an executable specification of the design.

The term TLM is somewhat fuzzy and usually refers to modeling at the cycle- and byte- accurate level. Raising the abstraction w.r.t data and timing accuracy can further extend the TLM paradigm.

This paper introduces a unique packet-level TLM style, for the conceptualization and system level performance analysis of a large heterogeneous system. Of course the design entry for small scale and homogeneous systems is still at Register Transfer Level (RTL) and medium scale embedded systems are best modeled at cycle accurate TLM.

B. *Functional Modeling*

In packet-level TLM the bit-true data representation of RTL is replaced by the Abstract Data Types (ADTs) such that a set of functionally associated data is represented as a single token (see Figure IV-2). The functional blocks communicate to each other through a point-to-point reactive channel. Since the system state changes only at the arrival of a new token the number of activations on the event-driven SystemC simulation kernel is reduced, to provide a high simulation speed.

C. *Architecture Exploration*

At this level the importance is given to the profiling of the bus-based communication architecture and its effect on the system performance. The point-to-point communication is replaced with the address-mapped shared memory data exchange as a natural choice. To achieve high simulation speed and modeling efficiency, we also model the shared bus channel at the packet-level. The functional blocks re-use the ADT interface and communicate to each other using a packet-level bus channel. The re-use of interface and the modeling efficiency decreases the turn-around time.

1) *Abstract Bus Channel- Generic Bus*

The more specific implementation details of the shared bus are suppressed at this level. The low level details like pin-accuracy, specific bit-width of the data ports and cycle accuracy are suppressed here. Instead of transferring a block of data (for e.g. a Packet Descriptor) through 32-bit data port, the ADT itself is transferred.

The shared memory associated with the abstract bus stores the ADTs instead of bits/bytes of the data. More than one master can be connected to this bus and the arbiter can implement the specific arbitration algorithms. Our arbiter implements a priority based arbitration scheme.

This Generic Bus doesn't model any real-bus specific features. Using this abstract model, the system designer gets an executable model to carry out a rough profiling of the system.

2) *Near Cycle Accurate Bus Channel- NCA Bus*

After the rough profiling, the abstract bus channel is parameterized to model the timing and data transfer properties of a cycle-level TLM bus. This makes the profiling of the system very close to the realistic profiling at cycle accurate level. The parameters can be defined for enabling the pipeline and burst transfers of the bus. This takes care of the timing annotation for the features that are bus specific. With this parameterization the communication exhibits timing accuracy close to a cycle accurate bus.

D. *HW/SW Integration*

In today's SoC designs, a large part of the system functionality is realized in SW. To complete the system level verification these SW functionalities need to be integrated and simulated on an Instruction Set Simulator (ISS) of the targeted processor.

The ISS integration requires a cycle accurate interface with the rest of the system. Also, the bus models and the memory interfaces should be cycle accurate. To interface the functional models with a real cycle accurate bus an adapter need to be developed. The adapter takes care of the conversion of the ADT to the bytes/words and assembling the ADT from the bytes/words at cycle accuracy. Using these adapters the abstract functional models can seamlessly interface to any cycle accurate bus.

IV. High Level Architecture - Abstract Bus Channel

In this section we will introduce the internal structure of our packet-level bus and then discuss the timing annotation capabilities of the generic and NCA bus model. After that we show the link to further refinement stages enabled by a adapter between the packet-level and cycle-level TLM.

A. *Timing Annotation Principle*

The Figure IV-1 illustrates the basic communication and timing annotation principle of the abstract bus channels.

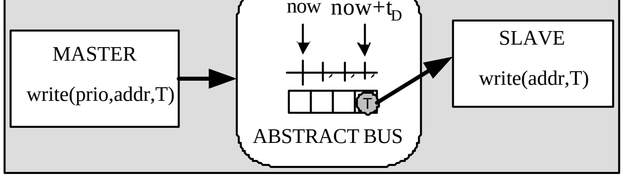

Figure IV-1: Bus channel behavior

The figure shows that, the master does a write transaction to the slave by calling the function write (prio, addr, T), where T represents the ADT. The bus channel automatically calculates the timing information t_D, associated with the data transfer.

After the time t_D, the bus channel calls the interface method, write (addr, T) of the slave and writes the ADT to the specified address.

```
class PacketDescriptor{
  int version;            -- pragma field 144, 8
  int ihl;                -- pragma filed 240, 8
  int tos;                -- pragma field 128, 8
  ...
  long dest_addr;      -- pragma field 64, 32
}
```

Figure IV-2: ADT - Sets of fields

The ADT'-s are stored in a shared memory and the ADT comprises different fields. Each of the fields is mapped to a bit-level position by specifying the address offset from the start address of that token and the bit-length of the field. The master flags the fields that will be transferred. The timing is calculated depending on the fields, which are flagged to be written/read and also depending on the size of the fields. For e.g. in Figure IV-2, the field version is located at 144th bit position of the bit-true ADT and has a length of 8 bits.

B. Timing Parameters

1) Timing calculation – Generic Bus

In the Generic Bus, the calculation of the timing information, t_D doesn't take care for the bus specific features. It helps the designer to derive a rough idea about the system level performance.

t_D = data transfer size/BW.

Where the data transfer size defines the total size of the data exchanged by the master from the enabled fields of the ADT and the BW is the bus bandwidth in bits.

2) Timing Annotation – NCA Bus

To model realistic transactions, the NCA bus channel is made configurable to mimic real-bus specific features, such as:

1. Internal Cycles: For e.g. to complete a data transfer (read/write), the bus may take one or more bus cycles depending on its timing specification.
2. Pipeline: If the system bus exhibits a pipeline behavior for the data transfer, the effect of internal cycles may not be visible for all the address & data phases. For e.g. AMBA bus.
3. Burst Mode: In systems, where we use the AMBA bus, the size of the data transfer and the burst length play an important role when we calculate the timing. Such effects are taken into account in the NCA bus.

With these configurable parameters the designer can model the communication timing without significant loss of accuracy.

In this sub-section we describe the algorithm to calculate the timing annotation value, t_D for NCA Bus. The bit-true analysis of the ADT has the following steps:

- Sort the enabled fields by the staring address of the fields
- Form blocks of adjacent fields
- Break the transactions to single or burst transfers based on a ddress alignment.

During the calculation of t_D, for the single transfers the full effect of the internal cycles are taken into consideration along with the pipeline. For the burst transactions the effect of internal cycles are not visible for all the address and data phases.

With this timing annotation we can accurately analyze the behaviors such as bus-contention, busload and arbitration effects, which will affect the system performance. The advantage of this NCA bus channel is that it gives higher simulation speed with less loss of accuracy compared to a real cycle accurate bus.

C. Cycle Accurate TLM Adapter

To integrate the SW functionality running on an ISS, an adapter is needed to seamlessly interface the functional models to the cycle accurate bus. The real cycle accurate bus used in our system is AMBA bus.

A cycle-level TLM bus, like AMBA bus, transfers the data (byte/words) at cycle accurate level. The Functional models, which exchange the ADTs cannot be directly interfaced to a cycle accurate bus. The functional models are connected to the cycle accurate bus through an adapter.

The system with the adapter is shown in the Figure IV-3. The functional models are reused without any changes. The adapter converts the ADT transactions to cycle-level TLM bus transactions (bytes/words) at cycle accurate level.

The adapter re-uses the NCA bus algorithm to decide the transaction features (burst length, single/burst transfer type), depending on the fields, which are marked for transfer.

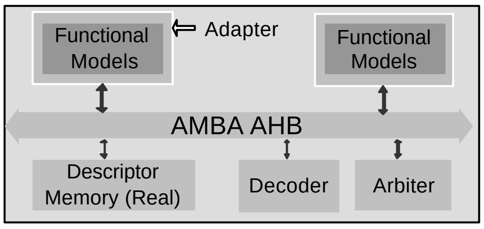

Figure IV-3: Architecture view with Cycle-level TLM bus

After the bit-true analysis of the ADT, the adapter breaks the data transfer into different AMBA bus transactions by calling the AHB bus interface methods with the proper transfer types and burst length.

V. CASE STUDY

In this section we show the results of the simulation run of an OC-48 (2.5Gbps) IP Forwarding design example with Quality of Service support (Figure IV-1). In this example the Descriptor Bus was replaced with abstract bus model and with the real cycle accurate AMBA bus during the different simulation runs.

This design example and test bench contains about 20,000 lines of C++ code having 10 modules, 18 point to point channels and 1 abstract bus channel. The traffic pattern to the test bench contained IP packets of size 40 bytes to 200 bytes.

The comparison will give the timing annotation inaccuracies in the abstract bus model and the simulation speed of the systems.

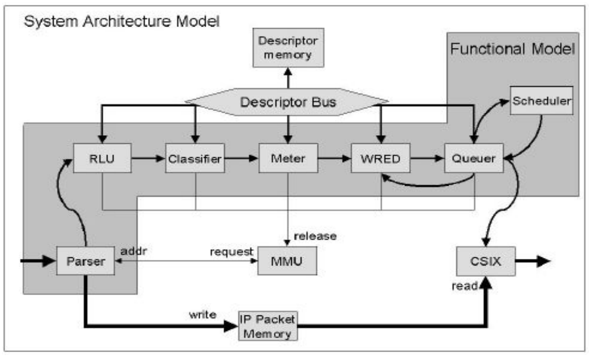

Figure V-1: IP Forwarding Chip

1) Timing Inaccuracy of the Abstract Bus channel

The Table V-1 shows the results of the simulation run with the different bus models. The table shows the average packet latency with respect to the IP Packet size. To minimize the effect of scheduling algorithms we use constant size packets into the same queue in the simulations. The average packet latency in cycles is used as the objective for the timing error calculation.

IP Packet Size	Generic Bus	NCA Bus	Cycle-level TLM bus
40	120	173	185
80	158	213	225
120	198	253	265
160	238	293	305
200	278	333	345

Table V-1: Average IP Packet forwarding latency

The NCA bus timing inaccuracy is within 12 cycles as it is parameterized to model the realistic bus effects. The generic bus shows huge inaccuracy due to unrealistic timing annotations.

The table shows the basic timing inaccuracy due to the specific bus timing annotations. As we simulate long sequences of IP packets, the average transactions on the bus increases and the effects of bus arbitration come into picture. At increased transactions on the bus, it is observed that the difference between NCA and cycle-level TLM bus system remains at 7% inaccuracy.

The relative inaccuracy between the NCA bus and cycle-level TLM bus system is due to the extra cycles for the synchronization between the master and AMBA bus and the impact of locked transactions on the AMBA bus. The synchronization and the locked transaction are absent in NCA bus.

2) Simulation Speed

Generic Bus	NCA Bus	Cycle-level TLM bus	Cycle-level TLM bus with ISS
100%	71.78%	43.68%	10.41%

From the simulation performance, it can be noticed that the NCA bus channel gives very high simulation performance compared to the cycle-level TLM bus system.

VI. Conclusion

In this paper a high efficient modeling of the bus-based communication architecture is introduced. The outlined modeling concept is capable of modeling realistic performance issues of the system without much loss of accuracy. The resulting modeling efficiency combined with high simulation speed addresses the communication architecture exploration for the complex SoC designs.

The outlined modeling style has been applied to the modeling and conceptualization of an IPv4 forwarding chip. The system level communication architecture is successfully refined from the generic abstract level to the cycle accurate level with the help of adapters. The resulting architecture platform is used to carry out the HW/SW integration as well, by integrating the Classifier functionality on ARM946-ES ISS [15] and also on LISATek processor simulators [16].

References

[1] M.Lajolo, et al., "A Case Study on Modeling Shared Memory Access Effects During Performance Analysis of HW/SW Systems", Int. Workshop on Hardware/Software Codesign (codes/CASHE), 1998.

[2] MESH, "Methodology for Executable Specifications Hierarchy", Intellectual Property and Design Services (Wireless & Broadband Communications), Synopsys Inc.

[3] Tim Kogel et al., "Virtual Architecture Mapping: A SystemC based methodology for architectural exploration of System-on-Chip designs", Int. Workshop on Systems, Architectures, MOdeling and Simulation 2003.

[4] DesignWare^{R} AMBA^{TM} SystemC^{TM} library User Guide, Synopsys Inc.

[5] VSI Alliance on-chip bus DWG. "On chip bus attributes specification" version v1.1.0 (http://www.vsi.org/library/specs/summary.htm)

[6] "ARM announces AMBA-SystemC interfaces to enable System Level Design" http://www.arm.com/

[7] The Core Connect^{TM} Bus Architecture, http://www-3.ibm.com/chips/techlib/techlib.nsf/productfamilies/CoreConnect_Bus_Architecture

[8] Lahiri et al., "Lottery Bus: A new High Performance communication architecture for System-on-Chip designs", Design Automation Conference, 2001.

[9] V. Lahtinen et al., "Interconnection scheme for continuous-media System-on-a-Chip", Microprocessors and Microsystems, Vol.26, Iss.3, April 2002.

[10] W. Cesario et al., "Component-Based Design Approach for Multicore SoCs", Design Automation Conference, 2002.

[11] S. Kumar et al., "A Network on Chip Architecture and Design Methodology", IEEE Computer Society Symposium on VLSI 2002

[12] L. Torres et al., "Bus analysis and performance evaluation on a SoC platform at the system level design", 11th IFIP International Conference on Very Large Scale Integration, 2001

[13] SystemC Initiative, http://www.systemc.org

[14] Thorsten Grotker et al., "System Design with SystemC", Kluver Academic Publishers, ISBN 1-4020-7072-1.

[15] DesignWare^{R} ARM^{R} Processors SystemC^{TM} library User Guide, Synopsys Inc.

[16] LISATek product family, http://www.coware.com/