ARTS: A System-Level Framework for Modeling MPSoC Components and Analysis of their Causality

Mahadevan, Shankar; Storgaard, Michael; Madsen, Jan; Virk, Kashif Munir

Published in:
13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)

Link to article, DOI:
10.1109/MASCOTS.2005.16

Publication date:
2005

Document Version
Publisher's PDF, also known as Version of record

ARTS: A System-Level Framework for Modeling MPSoC Components and Analysis of their Causality

Shankar Mahadevan[†] Michael Storgaard[‡] Jan Madsen[†] Kashif Virk[†]

Informatics and Mathematical Modelling (IMM), Technical University of Denmark (DTU), Richard Petersens Plads 2800 Lyngby, Denmark

E-mail: [†]{sm, jan, virk}@imm.dtu.dk, [‡]{s011934}@student.dtu.dk

Abstract

Designing complex heterogeneous multiprocessor System-on-Chip (MPSoC) requires support for modeling and analysis of the different layers i.e. application, operating system (OS) and platform architecture. This paper presents an abstract system-level modeling framework, called ARTS, to support the MPSoC designers in modeling the different layers and understanding their causalities. While others have developed tools for static analysis and modeled limited correlations (processor-memory or processor-communication), our model captures the impact of dynamic and unpredictable OS behaviour on processor, memory and communication performance. In particular, we focus on analyzing the impact of application mapping on the processor and memory utilization taking the on-chip communication latency into account. A case-study of a real-time multimedia application consisting of 114 tasks on a 6-processor platform for a hand-held terminal shows our frameworks co-exploration capabilities.

1. Introduction

A key pre-requisite in the design of heterogeneous multiprocessor system-on-chip (MPSoC) is an abstract system-level model that enables evaluation options and make critical architectural decisions in advance of a detailed design. The scheduling problem, central to the analysis of the complexity of concurrent MPSoC programs, depends on the way in which the tasks are mapped on the processing elements (PE). This, in turn, is linked with the physical architecture of the computing platforms, i.e. with task execution latency of the PEs, memory constraints of the PEs which limits the number of tasks mapped to a given PE, and the amount of data to be transferred between tasks mapped to different PE, which will influence communication latency and dynamic memory allocation. When scheduling is handled by a real-time operating system (RTOS) and not statically during compile-time, the system analysis becomes particularly challenging.

We propose an abstract system-level modeling framework, called ARTS, which captures the cross-layer dependencies of application software, RTOS, and multiprocessor platform consisting of PEs connected through an on-chip network. In this paper we focus on two issues of importance for analyzing cross-layer dependencies; static and dynamic memory usage and communication latency due to network topology and protocol. We illustrate the capabilities of ARTS for modeling and analyzing heterogeneous MPSoC systems, by exploring a real-time multimedia application consisting of 114 tasks on a 6-processor MPSoC architecture for a hand-held terminal.

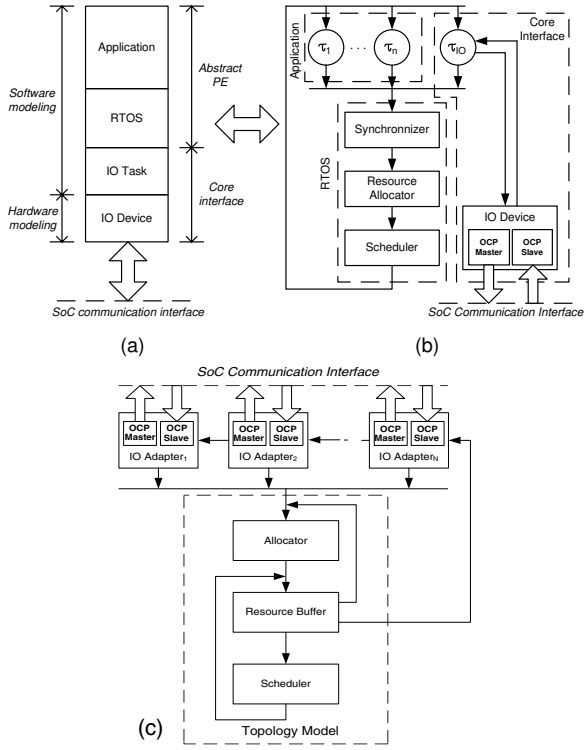

Figure 1. The PE Model: (a) Layer Structure (b) Block Diagram. (c) Network Model.

System-level models for design space exploration of embedded systems targeted for real-time applications has been proposed in [3, 8]. In [3], while supporting extensive RTOS capabilities to evaluate processor utilization, it does not address memory and communication concerns. Tools presented in [1, 4, 9] support processor-memory or processor-communication co-exploration and are important contribution in expanding the scope of design space exploration. In [4], the proposed framework is integrated with a two step

Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'05)
1526-7539/05 $20.00 © 2005 **IEEE**

co-exploration methodology of static-analysis followed by trace-driven simulation for design evaluation.

We do not propose any specific methodology for design exploration, but provide a flexible framework for designer driven exploration. Using our framework, algorithms such as energy-delay driven memory analysis [2] can be applied prior simulation to group and partition application tasks, or post-simulation traces collected such as those required to identify and tune platform tradeoff [7].

2. The ARTS Modeling Framework

This section presents the PE and application models, followed by details of the communication and memory extensions which are the focus of this paper. The models are implemented in SystemC.

2.1. PE and Application Model

In [5, 6], the PE and application model has shown to be sufficient to model the execution behaviour of a wide range of IP cores. In this paper, the model has been extended with a OCP (v2.0) based *core interface*, Figure 1(a) and (b), for inter-processor communication at RT and transaction levels.

The abstract Real-Time Operation System (RTOS) provides RTOS services, such as task *synchronization*, *allocation* of shared non-preemptive local resource between tasks and task *scheduling* for execution. Protocols supported are Direct Synchronization for the task synchronization, basic priority inheritance for the resource allocation and Rate Monotonic (RM) and Earliest Deadline First (EDF) for the task scheduling. The RTOS manages the tasks and its timing constrains, which is provided by the application model. The application model is based on static task graphs (or dataflow graph). For task modeling, a periodic and sporadic task model is available. Both models supports preemption.

The core interface consists of an IO task and IO device, modeling an IO device driver application and a hardware IO port respectively. The IO task manages the encoding/decoding of data to/from the SoC communication interface. The IO task is released for execution, whenever an inter-processor communication event starts (i.e. transmitting or receiving data). The IO device implements/manages the OCP SoC communication protocol.

2.2. Network model

The network model allows modeling of different communication topologies ranging from a single shared bus to a 1D/2D mesh NoC with minimal path routing. The model is characterized by having an abstract description of the topology but being able to support transmission of real data (e.g. at RTL). Further, it supports multithreaded out-of-order communication. Figure 1(c) shows a block diagram of the model.

The IO adapter model implements and manages the SoC communication protocol and the data conversion between the

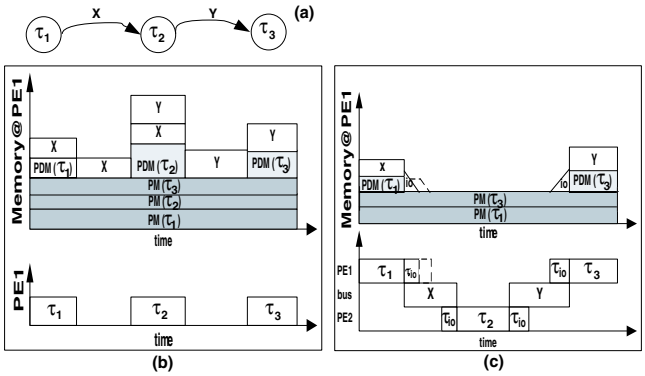

Figure 2. (a) Task graph. Memory profile of PE_1: (b) when all tasks run on PE_1, and (c) when τ_2 is mapped to different PE.

topology model and the SoC communication interface. When data is received from the SoC communication interface, it issues a process data package message to the topology model. Similar, when data package message is received from the topology model (indicating data has reached the destination node), it initiates a SoC transaction to the particular node.

The topology model describes the communication topology. It ensures that a data package message is not released to the destination IO adapter, until a time interval, equal to the communication latency, has expired. The *allocator* models the actual topology and manages the usage of shared communication resource (e.g. links and routers, bus). It assigns resources to the messages as they are received. The *resource buffer* models the resource usage mechanism, by buffering a data package before releasing it back to the allocator again. The interaction between the allocator and the resource buffer models the chain of communication tasks in the communication layer (i.e. the usage of different links and routers for a particular transfer). The *scheduler* models the scheduling of data package messages in case of resource contention. Messages assigned to an occupied resource gets buffered in the scheduler, until the resource becomes available. The current protocol implemented is first-come-first-served.

2.3. Memory Model

The memory model, models both static memory allocation, due to program memory (PM) and dynamic allocation, due to total data memory of the task. The example in Figure 2 illustrates the memory model. It shows the scheduling and resulting memory profile (spilt into static and dynamic memory). The dynamic part is split into private data memory (PDM) needed while executing the task and data memory needed to store data for exchange among tasks.

The total data memory size of the tasks, which is allocated during runtime by the RTOS, is calculated based on precedence constraints. We take a conservative view, i.e. during execution, the task data memory profile is the sum of preceding and succeeding data edges. This is observed for data x of PE_1 (Figure 2(b)) which, is created and dynamically allo-

Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'05)
1526-7539/05 $20.00 © 2005 IEEE

Applications	Tasks/ Edges	Deadline/ Period (ms)
GSM Encoder	53/80	20
GSM Encoder	34/55	20
MP3 Decoder	16/15	25
JPEG Encoder	6/5	500
JPEG Decoder	5/4	250

(a)

IP Cores	Frequency (MHz)
GPP0	25
GPP1	10
GPP2	6.6
FPGA	2.5
ASIC	2.5

(b)

Table 1. Application and IP Characteristics.

	PE#0	PE#1	PE#2	PE#3	PE#4	PE#5
Arch1: CT = 57006 μs, BC= 71						
IP	GPP0	GPP0	ASIC	GPP0	GPP0	ASIC
OS	RM	RM	-	RM	RM	-
Tasks	18	19	19	23	19	16
PEU (%)	100	59	47	23	61	24
Arch2: CT = 57568 μs, BC= 79						
IP	ASIC	FPGA	ASIC	FPGA	GPP0	ASIC
OS	-	RM	-	RM	RM	-
Tasks	18	19	19	23	19	16
PEU (%)	40	60	47	26	61	24
Arch3: CT = 58271 μs, BC= 70						
IP	GPP0	GPP0	ASIC	GPP0	GPP0	ASIC
OS	EDF	EDF	-	EDF	EDF	-
Tasks	18	19	19	23	19	16
PEU (%)	100	59	47	23	61	24
Arch4: CT = 58207 μs (Deadline Missed), BC= 97						
IP	GPP0	GPP0	ASIC	GPP0	GPP0	ASIC
OS	RM	RM	-	RM	RM	-
Tasks	19	9	24	24	15	23
PEU (%)	100	52	56	58	19	34

Table 2. Case Study Platform Architectures.

cated for the whole duration of task τ_1. As it has to be used by task τ_2, we have to keep the memory allocated until τ_2 has completed. After execution, only the succeeding data is saved, until the time where it is read by the succeeding task, or transferred to the NoC, after which it is deallocated.

Figure 2(c), shows a scenario with NoC transfer, requiring x and y to be transferred over the bus. τ_{io} emulates the device driver for the PEs and, dynamically allocates (y in PE_2) and deallocated (x in PE_1) the needed memory. If the IO task has to stall, i.e. due to bus congestions, the memory profile of the IO task will be a step function as illustrated with dashed lines in Figure 2(c). As the IO task is handled by the PE, any stall will result in an increased latency. Depending on OS scheduling, the time slot when data memory is initialized and deallocated has direct impact on network congestion.

3. Case Study

To illustrate the potential of our framework, we look at an embedded subsystem that executes GSM, JPEG and MP3 applications (Table 1(a)) - in all 114 tasks. Based on the PE choice of GPP, ASIC or FPGA (Table 1(b)); the tasks have different execution properties i.e. best- and worst-case task execution times. Further we can apply, RM or EDF scheduling to the RTOS. We experiment with different platforms, task partitioning and OS choice on a 6 PEs platform connected via a bus. Even for this simple platform, there are in all 15625 (5 IP cores characteristics applied to 6 PEs)

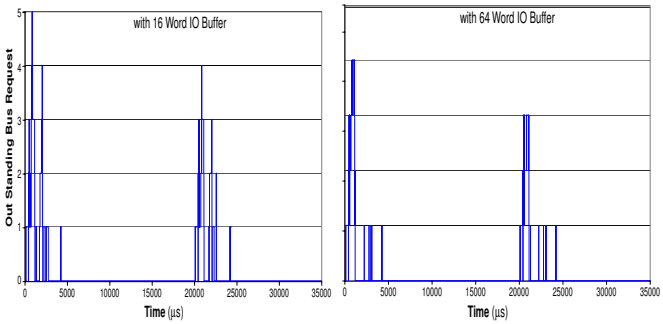

Figure 3. Bus Contention in Arch2.

possible architectural combinations, for a given partition and OS. Co-exploration in this multi-dimensional space, presents many suitable platform architectures and complex trade-off scenarios.

Table 2[1] shows four platform architecture instances, labeled from **Arch1** to **Arch4** from the co-exploration space. The changing characteristics within these platforms is either, IP cores, OS or task partitions. The Completion Time (CT) and the bus contention (BC) is also presented. Using the ARTS framework, we investigated these platform architectures in the context of modeling and understanding the causality between their system properties.

The correlation at $t = 0$ and $t = 20000\,\mu$s in BC and memory profile (Figure 3 and 4), is due to the GSM applications, which have a period of 20ms (Table 1(a)). However, the correlation is not identical and it depends on the OS applied to the platform architectures, and the task mapping. Following is the discussion of the system properties in additional detail.

PE Exploration: **Arch1** and **Arch2**, which use identical interconnect and OS in their platforms, present interesting trade-off of performance *vs* flexibility. **Arch1**, with four GPP0s, provides greater programming flexibility (software), while **Arch2**, with two FPGAs, provides greater flexibility in configurable hardware. The difference in CT and BC, among the architectures, is small (Table 2), however the PE utilization (PEU) of **Arch2** is well balanced among the IPs, compare to **Arch1**. This is due to the presence of ASIC which brings added performance.

OS Exploration: For a given platform architecture, a change of OS on one PE may have non-local consequences on other system components. This is due to management and scheduling of task executions by the RTOS, which in turn influences the causality, for example with bus access and memory spread. In **Arch1** and **Arch3**, the switch from RM to EDF, albeit presents limited effect on PEU or CT or BC, it does impact the peak bus occupation (3 in **Arc3** oppose to 5 in **Arch3**). In this case it is favourable to use EDF, since majority of bus contention in earlier architectures where due to conflict between GSM and JPEG, where JPEG transfers large streams of data over the bus, blocking GSM. Due to

[1]The simulation time with complete result logging (PE Utilization, bus contention and memory profiling), for one platform architecture was 0.15sec on Intel Pentium IV® 1.99 GHz with 512 MB of RAM.

Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'05)
1526-7539/05 $20.00 © 2005 **IEEE**

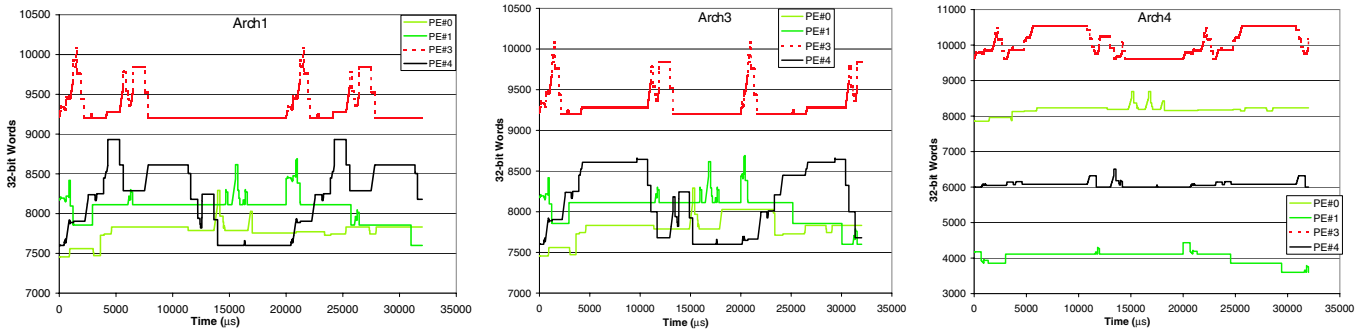

Figure 4. Memory Profile for GPP and FPGA PEs.

EDF, the conflicting JPEG task is swapped with local GSM task, which has higher priority due to its early deadline, there by reducing bus contention. The impact on memory spread can be seen in Figure 4.

Communication Exploration: Communication exploration can be undertaken at different granularity. For example at architectural level by using bus or multi-hop on-chip network topology model (Section 2.2), or at component level by changing the buffer size in the IO adapter. We modeled **Arch2** with IO buffer of size 16 words and then with 64 words, as seen in (Figure 3), which reduced the bus contention. The idea behind this experiment, is to show that even relatively minor tuning within one of the system components, could provide significant system-level gains, without needing deployment of "superior" alternatives, whose impact has not yet been ascertained.

Memory Exploration: The goal of updating the task partition, from that presented in **Arch1** to **Arch4** (Figure 4), was to reduce the peak memory usage of **Arch1**. However, the resulting architecture **Arch4** causes the MP3 to its deadline. It is because, the two task from concurrent branch of the MP3 Decoder are mapped on to the same PE. Interestingly, the partition shows a more balanced PEU, low bus contention and better peak-to-average memory usage, and thus may not be discarded lightly. Mapping the conflicting MP3 task to an alternate PE could potentially bring higher benefits compare to other architectures discussed so far.

The above explorations, shows that choosing an optimum platform architecture of a cross-layered complex design involves studying a large set of viable solution space.

4. Conclusions

The ARTS is a simulation-based framework for single-chip designers to model and explore complex MPSoC designs. In this paper, we have presented valuable extensions to this model by introduction of the communication model and a memory model. The versatility of quasi-static based application models, along with runtime independent execution model, combined with RTOS and communication platform, enables the ARTS framework to develop and explore a broad class of designs. This has been demonstrated in a co-exploration case study for multimedia applications typical in the hand-held device. We have shown various capability and features of our framework which allow selecting and tuning platform exploration under given system constraints. In future, we will extend the model to include dynamic power and area analysis, as additional parameters for trade-off analysis during MPSoC exploration.

References

[1] G. Braun, A. Wieferin, and A. Nohl. Processor/Memory Co-Exploration on multiple abstraction levels. In *Proceedings of Design, Automation and Testing in Europe Conference 2003 (DATE03)*. IEEE Computer Society, March 2003.

[2] W. Fornaciari, D. Sciuto, C. Silvano, and V. Zaccaria. Fast system-level exploration of memory architectures driven by energy-delay metrics. In *ISCAS*, 2001.

[3] S. Honda, T. Wakabayashi, H. Tomiyama, and H. Takada. RTOS-Centric hardware/software cosimulator for embedded system design. In *Second IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES ISSS)*, September 2004.

[4] S. Kim, C. Im, and S. Ha. Efficient exploration of On-Chip bus architectures and memory allocation. In *Second IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES ISSS)*, September 2004.

[5] J. Madsen, S. Mahadevan, and K. Virk. Network-centric, system-level model for multiprocessor system-on-chip simulation. In J. Nurmi, H. Tenhunen, J. Isoaho, and A. Jantsch, editors, *Interconnect-Centric Design for Advanced SoC and NoC*, chapter 13, pages –. Kluwer, 2004.

[6] J. Madsen, S. Mahadevan, K. Virk, and M. Gonzalez. Network-on-chip modeling for system-level multiprocessor simulation. In *Proceedings of the 24th IEEE International Real-Time Systems Symposium (RTSS03)*, pages 82–92, 2003.

[7] A. Maxiaguine, Y. Zhu, S. Chakraborty, and W.-F. Wong. Tuning SoC platforms for multimedia processing: Identifying limits and tradeoffs. In *Second IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES ISSS)*, September 2004.

[8] R. L. Moigne, O. Pasquier, and J.-P. Calvez. A generic RTOS model for real-time systems simulation with systemc. In *Proceedings of the conference on Design, automation and test in Europe*, page 30082. IEEE Computer Society, 2004.

[9] A. Wieferink, T. Kogel, R. Leupers, G. Ascheid, H. Meyr, G. Braun, and A. Nohl. A system level processor/communication co-exploration methodology for multiprocessor system-on-chip platforms. In *Proceedings of the Design, Automation and Test in Europe Conference (DATE'04)*, page 21256. IEEE Computer Society, 2004.

Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'05)
1526-7539/05 $20.00 © 2005 **IEEE**