IPSJ Transactions on System LSI Design Methodology Vol.5 44–54 (Feb. 2012)

[DOI: 10.2197/ipsjtsldm.5.44]

Regular Paper

A Fast Performance Estimation Framework for System-Level Design Space Exploration

Seiya Shibata[1,2,a)] Yuki Ando[1] Shinya Honda[1]
Hiroyuki Tomiyama[3] Hiroaki Takada[1]

Received: May 27, 2011, Revised: September 2, 2011,
Accepted: October 19, 2011, Released: February 21, 2012

Abstract: This paper presents a fast performance estimation framework and an performance estimation method for design space exploration at system level. As the complexity of embedded systems grows, design space exploration at a system level plays a more important role than before. In the system-level design, system designers start from describing functionalities of the system as processes and channels, and then decide mapping of them to various Processing Elements (PEs) including processors and dedicated hardware modules. A mapping decision is evaluated by simulation or FPGA-based prototyping. Designers iterate mapping and evaluation until all design requirements are met. In order to shorten the evaluation time, we have developed a fast design space exploration framework which combines our system-level design tool, named SystemBuilder, and a newly developed fast performance estimation tool, named SystemPerfEst. SystemPerfEst is based on trace-based simulation method. The trace is obtained as the result of SystemBuilder, and the trace is fed to SystemPerfEst smoothly. Since the estimation of a design candidate finishes in about one second, design space exploration of a number of design candidates can be performed with SystemPerfEst in a practical time. A case study on design space exploration of a JPEG decoder system demonstrates the effectiveness of our framework.

Keywords: system-level design, performance estimation, design space exploration

1. Introduction

In order to design embedded systems of high quality in a short time, fast and accurate evaluation is musts for design space exploration. As the complexity of embedded systems grows to the extent of MPSoCs (multiprocessor system on a chip), design space exploration at a system level plays a more important role than before. In the system-level design, system designers start from describing functionalities of the system as processes and channels which indicate computations and communications among processes, respectively. Then the designers decide mapping of them to various Processing Elements (PEs) including processors and dedicated hardware modules [10]. A mapping decision is evaluated regarding performance and costs by simulation or FPGA-based prototyping. The designers iterate mapping and evaluation until all design requirements are met.

In the multiprocessor system design, designers should explore and find a good design candidate which meets their requirements from a vast design space. In order not to miss the best mapping of a system, exhaustive exploration is ideal solution for design space exploration. However, performance evaluation, which is one of the most important evaluation for design space exploration, makes exhaustive exploration hard since it requires measurable amount of time for some kind of simulation or evaluation with FPGA-based prototypes. Therefore, fast performance evaluation technique is necessary.

Traditionally, various simulation methods and simulation tools are proposed. Although cycle-level simulation is promising technique which achieves high accuracy, it needs huge time to simulate a system and not applicable for exploration by iteration of simulation. There are fast simulation and performance estimation methods at a high level of abstraction for fast design space exploration. Trace-based simulations proposed in Refs. [11], [16], and so on use traces for estimating execution time of a system and abstract out details of execution in order to reduce estimation time. Although these abstract simulation methods can simulate/estimate performance of systems in short time, they need accurate traces. Moreover, these works do not mention how to obtain such traces.

In contrast, system-level design tools proposed recently uses FPGA-based performance evaluation. Such tools surveyed in Ref. [6] automatically synthesize implementation of FPGA-based prototypes from high-level description of systems. These FPGA-based methods are fast and accurate, and need not any traces for performance evaluation. However, exhaustive exploration of design candidates cannot be performed in a practical time since synthesis time of an FPGA-based prototype reaches one or several hours.

[1] Graduate School of Information Science, Nagoya University, Nagoya, Aichi 464–8603, Japan
[2] Japan Society for the Promotion of Science, Chiyoda, Tokyo 102–8472, Japan
[3] College of Science and Engineering, Ritsumeikan University, Kusatsu, Shiga 525–8577, Japan
[a)] shibata@ertl.jp

In order to incorporate accuracy of FPGA-based prototypes and speed of performance estimation, some of works provide integrated frameworks which combine automatic synthesis functionalities and fast performance estimation methods [3], [12]. These works provide overall design flow from high-level description of system to fast performance evaluation, including methods for obtaining performance models. Although their performance evaluation techniques are fast for design space exploration, they lack consideration of communication time among functions and therefore they are applicable to systems with limited architectures.

We propose a fast and easy-to-use integrated framework for design space exploration of multiprocessor systems developed on an FPGA, combining our system-level design tool, named SystemBuilder, and a newly developed trace-based performance estimation tool, named SystemPerfEst. SystemPerfEst uses traces obtained from FPGA-based prototypes which automatically synthesized by SystemBuilder. In our framework, a system designer describes functionalities of a system first. Next, the designer synthesize FPGA-based prototypes of two extreme mapping (all software implementation and all hardware one) and obtains traces of these two prototypes on a target FPGA. Also, using a simple system description provided by our framework, designers characterize the target FPGA. Then performance estimation of the system with other mappings can be done using the traces.

Characteristics of the target FPGA is used for estimating communication time among functionalities. Since communication time depends on FPGAs and RTOSs (or middlewares) running on it, they should be considered for accurate performance estimation. In other works which assume that there are databases which contain architecture characteristics such as Ref. [3], designers should develop such databases by hand and with detailed knowledge about the target FPGA and RTOSs. In contrast, designers can collect the characteristics of the FPGA with just a single synthesis and execution of the simple system description on the target FPGA in our framework. Therefore designers are not required detailed knowledge about the FPGA to develop such databases.

The estimation method of SystemPerfEst is fast, and the estimation results are accurate to the extent that they are comparable with evaluation results on FPGA-based prototypes. Note that the basic estimation mechanism used in SystemPerfEst is a traditional trace-based one like Refs. [12], [15], [16]. However, our method can consider communication times spent by RTOSs and interruption handlers, and therefore our method is more accurate and applicable to wider architecture including RTOSs.

Moreover, since the input for SystemBuilder and SystemPerfEst is common, designers can verify the exploration results on an FPGA, make feedback to the description of systems, and explore more mappings using SystemPerfEst again smoothly.

The contributions of this work are (1) integrated framework which combines a system-level design tool and a performance estimation tool, and (2) performance estimation method for multiprocessor systems whose results are comparable with evaluation results of FPGA-based prototypes, considering communication times spent by such as RTOSs and interruptions.

Since our performance estimation method uses profiles of FPGA-based prototypes, target platforms of systems are limited to FPGAs and cannot be applied to design of ASICs. However, recent growth of FPGAs on their speed and capacity is raising them to the level of industrial products. Therefore our method can be used for such cases. Moreover, our method at least can be applied for design of ASICs for the purpose to prune out the obviously insufficient design candidates at early phase of design.

The rest of this paper is organized as follows. First, Section 2 presents a brief overview of related works about performance evaluation techniques for system-level design space exploration. Next, Section 3 explains an overview of our framework and SystemBuilder. Section 4 describes performance estimation method used in SystemPerfEst, and Section 5 shows the effectiveness of our framework through a case study. Finally Section 6 concludes this paper with a summary.

2. Related Works

There are many approaches which provide efficient evaluation environments for design space exploration.

ARTS [11] and TAPES [16] are system-level performance estimation frameworks. ARTS is a framework for modeling and simulating MPSoCs. Given profiles of tasks to be executed on processing elements, ARTS simulates communications among tasks and calculates performance numbers. TAPES provides a retargetable simulation framework with a given profile of the system functionality. These frameworks assume that profiles of the system at a system level are given prior to their simulation, therefore the accuracy of their simulation depends on the accuracy of profiles. Moreover, the focuses of these works are on qualitative analysis such as scalability for multiprocessor systems, and accuracy of them were not mentioned.

As for FPGA-based approaches to system-level design, automatic synthesis tools [4], [7], [14] enabled designers to reduce time for obtaining FPGA-based prototypes. Our system-level design tool, named SystemBuilder, was developed for a similar objective to them [8]. These works are all supports system-level design space exploration by synthesizing FPGA-based prototype implementations from abstract models of systems. Although a lot of efforts of designers for evaluating various mappings are reduced by them, still system designers should wait completion of synthesis, and execute FPGA-based prototypes and record results of systems by hand, iteratively. It is time consuming and prevents designers from exhaustive exploration.

Some system-level design exploration frameworks which combine system-level automatic synthesis tools and fast performance estimation methods are proposed in Refs. [3], [12]. The approach proposed in Ref. [3] extracts functional characteristics from high-level models of systems, and explores a large number of design candidates at an implementation level considering the functional characteristics and hardware characteristics. Although their approach achieved design space exploration with high accuracy in a short time, it requires a database containing detailed hardware characteristics of target architecture to be developed. Moreover, their estimation method only considers logic level architecture inside a processor and a hardware module such as adders, and do

not consider multiprocessor architecture and their parallelism.

The approach proposed in Ref. [12] provides performance estimation method where performance parameters are obtained from execution results of Instruction set simulators (ISSs). Their approach considers multiprocessor systems and calculates accurate execution time of systems. However, the use of ISSs causes relatively slow estimation speed, and they do not consider communication time among processors.

The basic concept of our performance estimation method is similar to the work by Ueda et al. [15]. However, the focus of their approach is comparison of performance on different bus topologies with given IP components. Execution times of IP components are assumed to be given by IP database and to be constant. Therefore, their approach cannot accurately estimate performance of systems where some of IPs are activated several times and their execution times vary on each of their activation. Also, they do not mention about how to develop the IP database.

Moreover, most of these performance estimation methods do not consider RTOSs (scheduling times) and interrupts (interruption handling times) which are often used for developing embedded software.

Regarding the framework construction, our framework is similar to the works in Refs. [3], [12], developed by combining an automatic synthesis tool and a fast performance estimation method. Unlike above tools, SystemPerfEst can estimate performance of multiprocessor systems including dedicated hardware modules with short estimation time and high accuracy, considering scheduling time of RTOSs and interruption handling times.

3. Overview of Our Framework

3.1 Design Flow

Figure 1 shows the overall design flow. First, designers describe the functionalities of systems ("Functional description" in Fig. 1), and synthesize FPGA-based prototypes using our system-level design tool, named "SystemBuilder."

Next, designers obtain a few "Profiles on FPGA" from FPGA-based prototypes. Typically, profiles of all software implementation and all hardware one are obtained. All software implementation indicates mapping which all processes and channels are mapped onto a single processor. All hardware implementation indicates mapping which all processes excluding ones which can be mapped only on software are mapped onto hardware. Designers can obtain these profiles by specifying the two mappings in "Mapping Specification (for traces)," synthesize implementation of them using SystemBuilder and execute them on an FPGA.

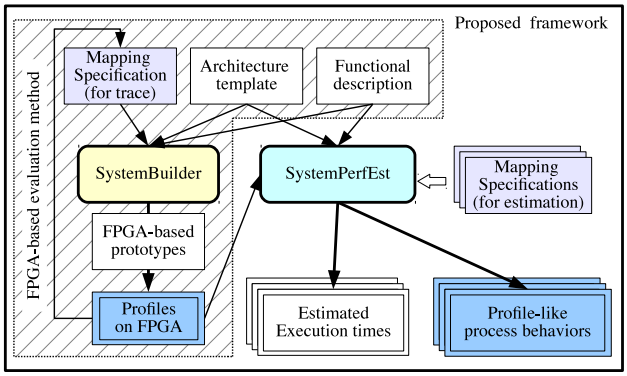


Fig. 1 Overall design flow of our framework.

After that, designers explore large number of mappings, which are specified using "Mapping Specification (for estimation)," with our performance estimation tool, named "SystemPerfEst." Since estimation of a mapping with SystemPerfEst finishes in seconds, designers can explore large number of mappings in hours. If the number of mappings is less than several thousands, designers can explore all of them using SystemPerfEst. Otherwise, designers can interactively explore mappings. The interactive exploration starts from estimating a subset of all mappings using SystemPerfEst. As a result of estimation, SystemPerfEst outputs not only execution time of mappings but also profiles like ones obtained from FPGA-based prototypes (described in Section 3.3) so that designers can analyze behavior of the system with them. Then designers estimate performance of next subset of mappings to be estimated, until a mapping which meets designer's requirements is found.

The strategy for efficient exploration is our future work and we leave it out of scope in this paper.

3.2 SystemBuilder

Here, we briefly explain about SystemBuilder [8] in order to make this paper self-contained.

Figure 2 shows the design flow achieved by SystemBuilder. SystemBuilder supports design space exploration of embedded systems by automatically synthesizing FPGA-based prototypes from functional description and mapping specifications.

Functional description for SystemBuilder consists of processes and channels. Processes represent computations, which can be mapped and implemented onto processors and hardware modules. Processes are described in the C language, and the channels are accessed from processes through APIs which is provided by SystemBuilder. Channels represent communications among processes and implemented on software and hardware depending on mapping of processes. Currently, SystemBuilder provides mainly two types of channels, FIFO channels and memory channels.

FIFO channels can be used for describing data/control dependencies between two processes. For instance, a receiver process of a FIFO channel is forced to wait until a corresponding sender process writes data to the channel. Memory channels represent storage of data transferred among processes. Typically, FIFO channels and memory channels are combined to represent FIFOs which transfer large data among processes. In the combination, FIFO channels are used for controlling accesses, and memory channels are used for buffers for data.

From inputs above, SystemBuilder automatically synthesizes descriptions of interconnections among processes. The synthesized communication descriptions are in the C language and VHDL, depending on mapping of the processes and channels. Also, SystemBuilder makes use of a cross-compiler of the processors for software and a behavioral synthesis tool for hardware module in order to obtain an executable binary and synthesizable RTL circuits, respectively. Processes mapped on software are compiled and linked with a Real-Time OS (RTOS). A FIFO

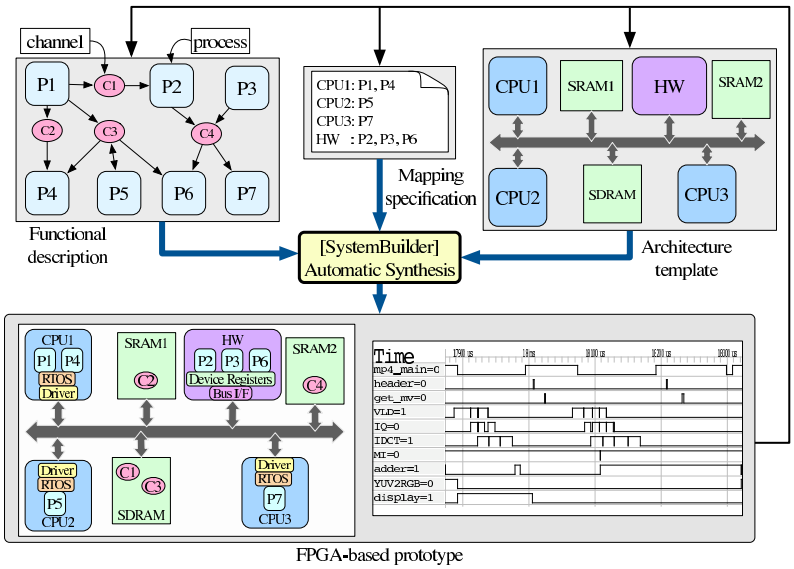


Fig. 2 Design flow of SystemBuilder.

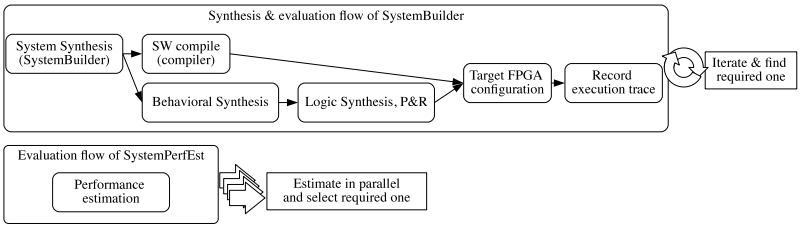


Fig. 3 Comparison of evaluation time of SystemPerfEst with FPGA-based evaluation method.

channel is transformed to a FIFO hardware buffer with circuitry to interrupt processors or a queue API of an RTOS by SystemBuilder depending on mapping of the processes. Memory channels are transformed to either of memory modules on an FPGA or arrays of the C language.

Finally, a configuration bitstream of the designed hardware architecture for an FPGA is synthesized by a logic synthesis tool from the RTL circuits and IPs such as processors and essential peripherals.

3.3 Process Profiler

SystemBuilder provides a process profiler, which is automatically configured and instrumented into FPGA-based prototypes [13]. We explain brief overview of the process profiler since our framework uses results of the process profiler.

The original purpose of the process profiler is to help designers see the behavior of processes on an FPGA and help them figure out mappings to improve the system. Generally speaking, on development of systems with complex and parallel process structure, it is hard to imagine execution orders and timings of processes considering their mapping, parallelism and data dependencies. In order to help designers understand the execution orders and timings of processes, the process profiler obtains them on FPGA-based prototypes and visualizes them.

In detail, the process profiler records a trace of activation/wait timings of processes through the execution period specified by a designer. Activation of processes means that the processes are computing between their FIFO channel API calls, and wait of processes means that processes are accessing channels.

Since processes are mapped onto either software or hardware, the process profiler records traces of all processes in a common timeline. The profiles can be shown as waveforms (shown in Fig. 2).

SystemPerfEst takes the results of the process profiler (process profile) as input traces to obtain execution time of processes. Also, SystemPerfEst outputs profiles like process profiles for interactive exploration using them.

3.4 Advantage of Our Methodology

Figure 3 briefly shows an advantage on design space exploration of our framework over a traditional exploration methodology with SystemBuilder (FPGA-based evaluation method).

In FPGA-based evaluation method, designers iterate evaluation by synthesizing and executing FPGA-based prototypes for a number of mappings. In detail, an evaluation consists of six steps; system synthesis by SystemBuilder, software compile, behavioral synthesis, logic synthesis and P&R, FPGA configuration, and recording execution traces. Although part of these steps are independent, each step needs high computation power and they cannot be done in parallel practically on a host PC. Hence even a single evaluation of a mapping spends long time. It is therefore hard to perform design space exploration by iterating evaluation of a number of mappings in practical time.

In contrast, our framework only uses SystemBuilder at first. Once the profile of all software implementation and all hardware one is obtained, designers can explore a number of mappings by

parallel execution of SystemPerfEst. Since SystemPerfEst is fast and can be executed in parallel on a host PC, design space exploration can be performed much faster than the FPGA-based evaluation method.

4. Fast Performance Estimation Method

This section describes detail of our performance estimation method used in SystemPerfEst.

4.1 Approach and Assumptions

In order to estimate the performance (total execution time) of systems fast and accurately, execution time of processes is the most important factor. Although the use of detailed simulation enables us to obtain them accurately, it will make estimation slow. Therefore, we abstract out computation part of process from process models and use execution time obtained from traces instead. Assuming that execution time of a process for an input on a PE does not change depending on mapping of other processes, execution time of the process will be the same as that in the trace of other mappings where the process is mapped on the same PE. Although this approach can be applied for estimating performance of a system on various mappings, it should be noted that this approach can be applied only for the same inputs and the same amount of computation as the input traces are obtained. Also, we assume that all PEs are in a single clock domain and are driven in a same clock frequency.

Under the assumption above, if there are n kind of PEs, n traces are necessary and sufficient for performance estimation. For instance, in the design space exploration is done on the architecture where processes can be mapped onto one type of processors and dedicated hardware ($n = 2$), performance estimation needs only two profiles. One is a profile of mapping where all processes are mapped onto the processor, and another is a profile of mapping where all processes are mapped onto dedicated hardware. The usage of them are described in Sections 4.2, 4.5 and 4.6. Obviously, our performance estimation method needs traces of processes obtained from FPGA-based prototypes generated by SystemBuilder or such system-level design tools. On the other hand, our method can be applied to systems targeting any FPGA with any processor if traces of processes can be obtained on them.

To make this assumption practically true, we assume that target processors have a sufficiently fast memory for their instructions and do not use any cache since the use of cache will make execution time of processors vary (described in Section 4.4).

In contrast, communication time among processes, especially time spent for blocking communications using FIFO channels, cannot be estimated simply using traces because of two factors. One is the implementation of channel APIs which depends on mapping of processes. Another is blocking time which changes depending on execution condition of a system.

Channel APIs, which are interfaces to channels, are implemented as RTOS API calls for software or as communication circuitry for hardware by SystemBuilder. Typically, a channel API call spends a constant amount of time depending on its implementation. Therefore, time for channel API calls can be estimated using a database which stores time spent for channels. Such database can be developed before exploration (described in Section 4.8).

On the other hand, blocking time cannot be estimated from traces since it depends on execution condition of other processes and channels. Therefore, performance models in our estimation method manages the occupation of FIFO buffers and blocking time (described in Section 4.5).

4.2 Concept of Our Estimation Method

Before going details, we show a concept of our performance estimation method.

Figure 4 illustrates the concept. Our method is a kind of trace-based simulation. We use two profiles obtained from FPGA-based prototypes as traces, one is a profile of all software implementation, and the other is that of all hardware implementation. From the two profiles, execution time (clock cycles) of processes on each computation step as software and hardware is obtained. According to mapping specification to be estimated, our estimation method selects profiles from all software implementation or all hardware implementation. After the selection of profiles, it calculates overall execution time (clock cycles) of the system considering parallelism of processes.

If the clock frequency of the system is determined, designers can calculate execution time (seconds) of the mapping by dividing the number of estimated clock cycles by the clock frequency. Although maximum clock frequency of a system generally may change depending on mappings, we leave the estimation method of them out of scope in this paper.

4.3 Overall Estimation Flow

Next, we illustrate estimation flow focusing on inputs and outputs in **Fig. 5**. SystemPerfEst takes six inputs: (a) functional structure of a system, (b) channel record, (c)(d) profiles of two mappings (all software implementation and all hardware one)

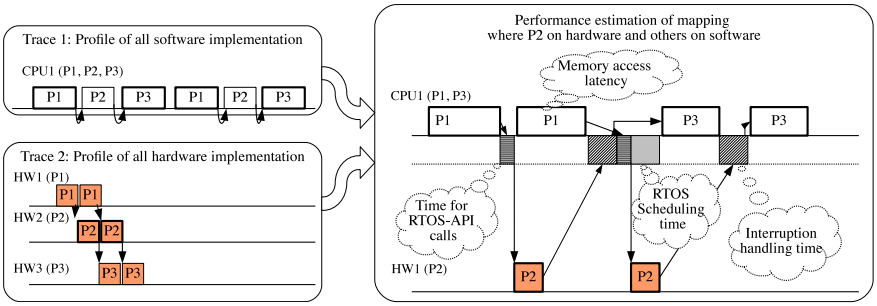


Fig. 4 Concept of our performance estimation method.

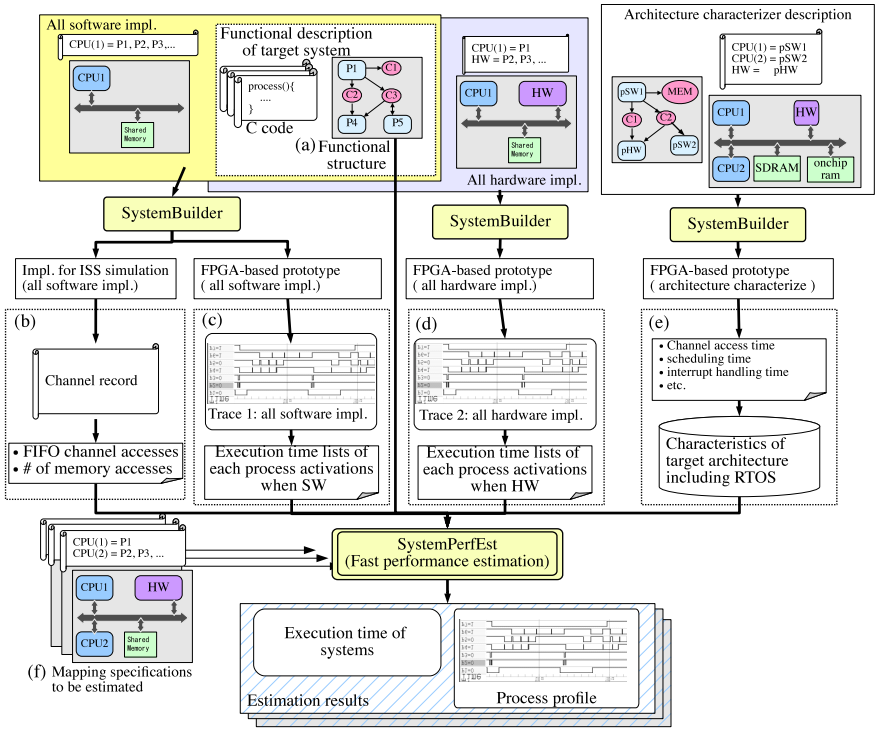


Fig. 5 Inputs and outputs of SystemPerfEst.

obtained by executing FPGA-based prototypes, (e) architecture characteristics, and (f) mapping specifications to be estimated.

Functional structure, profiles of a system and mapping specifications ((a), (c), (d) and (f)) are inputs and results of design flow of SystemBuilder. Therefore they are just fed to SystemPerfEst without any modifications.

Channel record (b) is a kind of event list which records invocation of channels by processes. Since process profiles do not contain records of channel invocations, SystemPerfEst needs them in order to know dependencies among processes and the number of memory accesses, which are determined in execution time of systems depending on its inputs. Channel record can be obtained from ISSs executing all software implementation of the system.

Architecture characteristics (e) are obtained from a profile of an FPGA-based prototype of a simple system description. The detail of architecture characteristics is described in Section 4.8.

As to the five inputs ((a), (b), (c), (d) and (e)), designers need to prepare only once at the first time of design space exploration of a system on a target FPGA.

With the six inputs, SystemPerfEst estimates performance of a design candidate described in mapping specifications (mappings). As the result of performance estimation, designers obtain execution time of systems and profiles which are in the same format as those of the process profile obtained from FPGA-based prototypes designed by SystemBuilder. After estimation of a number of mappings, designers can choose mappings which meet their requirements and/or analyze bottlenecks with the estimated profiles.

4.4 Target Architecture

Figure 6 shows a target architecture which SystemPerfEst can deal with. A single kind of processors is assumed. Under this assumption, system designers only need two traces (all software implementation and all hardware one).

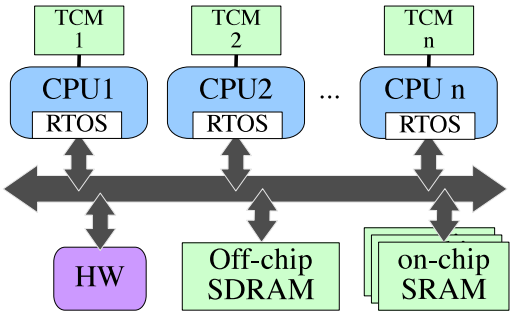


Fig. 6 Target architecture which our performance estimation method can be applied.

The number of processors and dedicated hardware modules where processes are mapped can be changed and explored. Processors are assumed to have their own TCMs (tightly coupled memories) for instruction/data of software and have no cache. This is because of our assumption described in Section 4.1. Consideration of caches is one of our future works.

A single clock frequency among processors and hardware modules is assumed.

Designers can also explore memory modules used in their system such as on-chip SRAMs and off-chip SDRAMs which vary latencies of them.

Currently, consideration of buses such as bus protocols and bus topologies is not included. In our estimation method, we assume architectures which use a single virtual bus where neither burst transaction nor memory/bus conflict occurs. However, effects of bus latencies on performance are included in memory latencies obtained from architecture characterization results (described in Section 4.8).

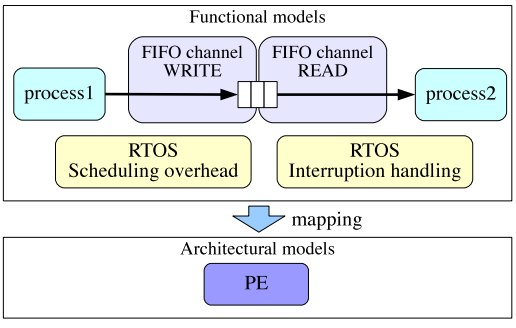
4.5 Performance Models

Figure 7 (a) shows performance models which SystemPerfEst defines. There are two groups of models, functional models and architectural models. Functional models represent processes, channels and RTOSs which are executed on PEs. Architectural models represent PEs which are used by functional models as shared resources. This separation of models is similar to Refs. [12], [15].

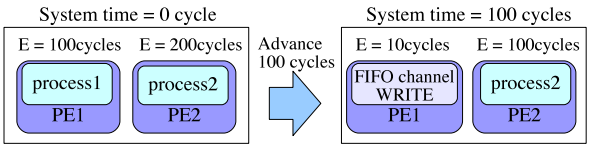
All functional models have their execution time list and two states, active state and wait state. In active state, they consume their execution time (that is, top of execution time list) when they obtain the right to use PEs they mapped onto. When their execution time becomes zero, they delete the top of execution time list and change into wait state. In wait state, they do nothing but wait for events to change into active state.

A process model is a functional model which represent a process in functional description of a system. Execution time list of processes is an input trace of the process selected from profiles according to their mapping. A completion of an active state activate corresponding FIFO channel model. Since FIFO channels which should be activated vary depending on inputs for a system, process models also need FIFO channel call list which are obtained from channel record. In a wait state, process models wait completion of FIFO channel access.

A functional model of a FIFO channel (a FIFO channel model) manages API call time and buffer occupation. FIFO channel models consume their API call time when they are on an active state. The completion of an active state means completion of channel access and activates corresponding processes. Execution time of active state of FIFO channel models is determined using architecture characteristics. In a wait state, FIFO channel models wait invocation of them by process models. The number of buffers can be set and explored by designers. In multiprocessor systems, API call time for send and receive may occur on different processors on inter-processor communications. We therefore made FIFO channel apart to two parts, send part and receive part (shown as "FIFO channel WRITE" and "FIFO channel READ" in Fig. 7 (a)).



(a) Performance models defined in our performance estimation method.



(b) Simple example of performance estimation with performance models.

Fig. 7 Performance models and estimation method using them.

There are two types of RTOS models, "Scheduling overhead" model and "Interruption handling" model. They manages scheduling overhead of RTOSs and interruption handling time used for FIFO channel between software and hardware, respectively. Execution times of them are determined by architecture characteristics. Both of them only time consumption on their PEs. Scheduling overhead model is activated when a process model on a PE changes. Interruption handling model is activated when a process on a processor is activated by a process on hardware after blocking.

PE models are architectural models which manage processes which can consume their time. In other words, PEs schedules functional models. PE models acts like OSs and have their scheduling policies. Preemption is also supported.

For instance, a PE which represents a processor have a priority queue which stores executable processes and channels, and selects one along with scheduling policy. Currently, SystemPerfEst supports fixed priority-based scheduling which have been mostly used in RTOSs for embedded system. Exploration of scheduling policies is one of our future works.

In this sense, functional models are similar to tasks on RTOSs and therefore they have priorities. Priorities of process models can be set and explored by designers. By default, all process models have same priorities. As for FIFO channel models and RTOS models, they have highest priority in order to simulate scheduling of RTOSs.

4.6 Performance Estimation Method

On performance estimation, SystemPerfEst manages system execution time which starts from zero, and increments along with process execution time obtained from profiles. Functional models consume their execution time only when they could obtain the right to use their PEs. When an execution time of a functional model become zero, its state changes to wait state and activates related functional models. After the iteration of this, the estimation ends with the system time at the moment as the resulting estimated execution time, when all execution time lists of functional models are consumed.

Figure 7 (b) illustrates an example. In the figure, there are two PE models ("PE1" and "PE2"). First, the system time is zero. At that time, process models "process1" and "process2" have the right to use the PEs (shown in left side of the figure). Their execution time are 100 cycles and 200 cycles, respectively. Then SystemPerfEst increments system time 100 cycles (minimum of execution times of functional models), and processes consume 100 cycles of their execution time. Then process1 lose the right to use PE1 and become wait state. At the same time, next functional model "FIFO channel WRITE" obtains the right to use PE1 as a result of scheduling of PE1.

4.7 Reflection of Memory Access Latencies

In order to explore memory mapping, SystemPerfEst can reflect changes of memory access latency. In input profiles, traces of processes include memory access time on a certain memory mapping. Therefore SystemPerfEst changes execution time of process obtained from traces when designers try to estimate per-

formance with different memory mapping.

Assuming that memory access latency to a memory module is constant, we can obtain execution time of process with a new memory mapping E_{new} from that with memory mapping in input profile E_{input} by $E_{new} = E_{input} + (L_{new} - L_{input}) \times N$. L_{new} and L_{input} represent latency of a new memory module and memory module used on input profile, respectively. N indicates the number of accesses to the memory region mapped on the memory module.

4.8 Architecture Characterization

SystemPerfEst considers communication time among processors and memory latencies. Such communication time depends on characteristics of a target FPGA and an RTOS executed on it. In order to take such characteristics into account, our framework provides a description of a simple system for architecture characterization (hereafter, architecture characterizer description). The description consists of some processes and channels and their mapping. By only synthesizing and profiling the description on a target FPGA using SystemBuilder, designers can obtain the database of characteristics of the target FPGA easily (shown as (e) in Fig. 5).

A structure of functional description and their mapping is shown in **Fig. 8**. Mapping of processes and channels are fixed. Channels in the system are implemented differently by SystemBuilder with the mapping. For instance, a channel between processes "cpu1_writer" and "cpu1_reader" is implemented as an RTOS API for inner-processor communication, while a channel between processes "cpu1_writer" and "hw_reader" is implemented as communication between software and hardware.

These processes also access memories in a target FPGA, and collect latencies of them. The collected memory latencies include latencies of buses among PEs and memories.

With the architecture characterizer system, designers can make database which contains following time; inter-process communication time (inner-processor, inter-processor and processor to HW), memory access latencies (an off-chip SDRAM, an on-chip SRAM) and RTOS time (scheduling overhead, interruption handling).

One of good example of advantage of architecture characterization is estimation of systems with different RTOSs from systems as input. For example, SystemBuilder generates TOPPERS/JSP kernel [2], which is one of the most popular RTOS in Japan, for single processor systems. On the other hand, as for multiprocessor architecture, SystemBuilder generates TOPPERS/FDMP kernel [2], which is a derivative of TOPPERS/JSP kernel for multiprocessor systems.

Since implementation details of communication APIs provided by the two RTOSs are different, communication time consumed by the APIs also differ. With architecture characterization, SystemPerfEst accurately estimate communication time for both systems on single processor architecture and multiprocessor architecture.

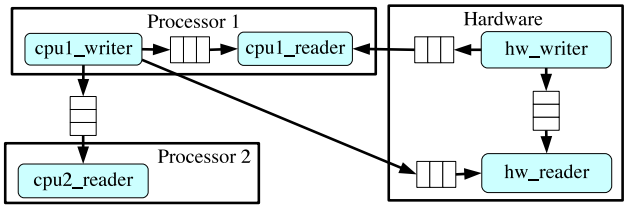


Fig. 8 Functional structure and mapping of the architecture characterizer description.

4.9 Discussions

4.9.1 Inputs for Systems in Design Space Exploration

SystemPerfEst can only estimate performance of system whose input is same as the trace is recorded. This limitation can be taken as one of disadvantage of the method. For example, worst-case performance cannot be estimated without worst-case input. However, we believe that this limitation is not crucial problem since worst-case input may be assumed and prepared in design requirements in the embedded system design.

4.9.2 Factors Not Considered

Currently, SystemPerfEst does not consider caches and arbitration delays on memory/bus accesses (memory/bus conflicts). The lack of consideration of them may cause huge estimation errors for systems with small caches and/or simultaneous access to memory modules. Consideration of these factors is of the most important work currently.

One solution for considering memory/bus conflicts for shared memory access may be the use of the method proposed by Kawahara et al. [9].

5. A Case Study on JPEG Decoder System Design

In order to demonstrate the effectiveness of our performance estimation framework, we show a case study of JPEG decoder system design.

The case study was performed on the following environment. The systems were designed using SystemBuilder on a PC whose OS is Windows XP Professional with an Intel Core 2 Quad 2.66 GHz processor and 2 GB RAM. The target board has an Altera Stratix II FPGA with Nios II soft-core processors at 50 MHz of clock frequency. eXCite 3.2c [5] was used for behavioral synthesis. Logic synthesis and P&R were done by Quartus 8.1. Performance estimation was performed on a PC whose OS is Linux with an Intel Xeon 2.93 GHz processor with 8 logical processor cores. SystemPerfEst is implemented in Python programming language and executed using psyco [1] which is a JIT compiler for Python.

5.1 JPEG Decoder Design Space Exploration

First, we designed a JPEG decoder system (**Fig. 9**). In Fig. 9, rectangles represent processes and processes with thick border can be mapped onto both processors and dedicated hardware. Arrows among processes represent FIFOs, which consist of FIFO channels and memory channels (as described in Section 3.2). For example, an arrow between "IQ" and "IDCT_pre" represent a FIFO which transfers 8×8 pixels of data with a buffer, Each FIFO shown in the JPEG decoder has three buffers. Process "top" and "main" can be implemented onto software only. Processes in the JPEG decoder can run in parallel in a pipelined manner. We evaluated performance of the JPEG decoder with an input image data

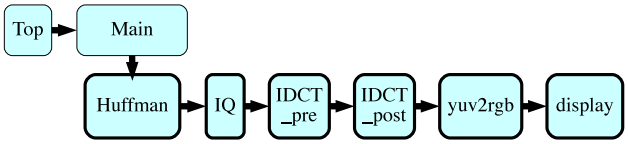


Fig. 9 Functional structure of the JPEG decoder system.

mapping	Huffman	IQ	IDCT_pre	IDCT_post	yuv2rgb	display
2core-pipe-1	1	2	2	2	2	2
2core-pipe-2	1	1	2	2	2	2
2core-pipe-3	1	1	1	2	2	2
2core-pipe-4	1	1	1	1	2	2
2core-pipe-5	1	1	1	1	1	2
2core-zigzag	1	2	1	2	1	2
3core-pipe-1	1	2	2	**3**	**3**	**3**
3core-pipe-2	1	2	2	2	**3**	**3**
3core-pipe-3	1	2	2	2	2	**3**
3core-pipe-4	1	1	2	**3**	**3**	**3**
3core-pipe-5	1	1	2	2	**3**	**3**
3core-pipe-6	1	1	2	2	2	**3**
3core-zigzag	1	2	**3**	1	2	**3**
1core-hw-1	HW	1	HW	1	HW	1
1core-hw-2	HW	1	1	1	HW	HW
1core-hw-3	HW	1	1	1	HW	1
1core-hw-4	1	HW	HW	HW	1	1
2core-hw-1	1	2	2	2	HW	1
2core-hw-2	HW	2	2	2	1	1
2core-hw-3	1	2	2	2	HW	HW
2core-hw-4	1	1	2	2	HW	HW
2core-hw-5	HW	2	2	2	1	HW

Fig. 10 Explored mapping of the JPEG decoder system.

in QVGA size (320 × 240 pixels).

Then we synthesized two FPGA-based prototypes of JPEG decoder (all software implementation and all hardware one) with non-preemptive scheduling policy, and obtained profiles of them using SystemBuilder. Note that all memory channels used for FIFO buffers are mapped onto an on-chip SRAM and fixed in this exploration. Since the on-chip SRAM is fast, effects of memory conflicts are supposed to be negligibly small. Also, we synthesized an architecture characterizer description twice in order to obtain communication time on the FPGA with RTOSs for single processor systems (TOPPERS/JSP kernel) and for multiprocessor systems (TOPPERS/FDMP kernel).

After that, we explored 22 mappings of the JPEG decoder (shown in **Fig. 10**) on target architecture shown in Fig. 6 using both our framework and FPGA-based evaluation method with SystemBuilder, and compared them. In the figure, "1", "2", "3" indicate processor number, and "HW" means hardware. For example, mapping "2core-hw-1" is a mapping where processes "Huffman" and "display" are mapped onto processor 1, processes "IQ", "IDCT_pre", "IDCT_post" are mapped onto processor 2 and process "yuv2rgb" is mapped onto hardware.

These mappings are selected in order to show that SystemPerfEst can estimate performance of mappings with various characteristics accurately. Since processes of the JPEG decoder system are executed in a parallel and a pipelined manner, typical cases of their mapping are separating them into former part ("Huffman" side part) and latter part ("display" side part), for utilizing their parallelism. Mappings of "2core-pipe-*" and "3core-pipe-*" are such typical cases. However, these mappings have only a single or two communications between processors (inter-processor communications) at separation point. In order to show the accuracy of estimation for mappings where many inter-processor communications occur, we also show the estimation results of mappings "2core-zigzag" and "3core-zigzag." Communications

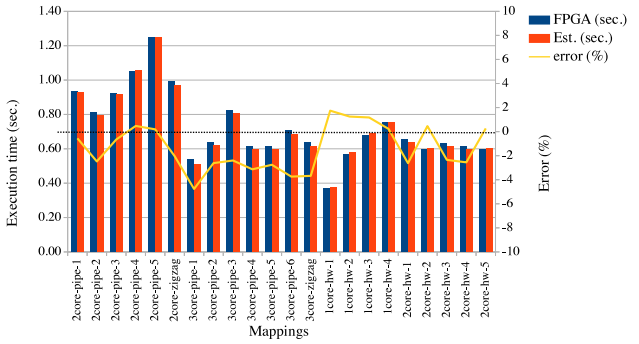


Fig. 11 Performance estimation results of the JPEG decoder system using on-chip SRAM for FIFO buffers.

between any two processes in these mappings are inter-processor communications. These two mappings are also typical cases which cannot utilize the parallelism of pipeline.

On mappings where some processes are mapped onto hardware, interrupt handling time has a large effect on their performance. Therefore, our performance estimation method should also consider the time correctly. In the mapping of "1core-hw-1," processes are mapped onto a processor and hardware alternately and therefore most of their communications are software-hardware communications which use interruption. In contrast, the mapping of "1core-hw-2" have less software-hardware communications than "1core-hw-1." Moreover, the mapping "1core-hw-3" have no hardware-hardware communication but the mapping "1core-hw-4" have two hardware communications. The mapping "2core-hw-*" shows the accuracy with two processors and hardware modules.

In summary, the 22 mappings represent typical cases sufficient for demonstrating the accuracy of SystemPerfEst.

5.2 Accuracy

Figure 11 shows a comparison between measured execution times of FPGA-based prototypes synthesized by SystemBuilder and estimated execution times. In the figure, blue bars in left side of two bars show execution time of mappings obtained from FPGA-based prototypes, while orange bars in right side show estimated execution time by SystemPerfEst.

Mean absolute error (MAE) of estimation results was 1.92%, distributing from −4.77% to 1.74%, which is thought to be sufficient for comparative evaluation of mappings.

One of the reason of these good estimation results is architecture characteristics which is obtained for both RTOSs for single processor systems (single processor RTOS characteristics) and multiprocessor systems (multiprocessor RTOS characteristics), as discussed in Section 4.8. As an example, we show a comparison of the performance estimation results of 2 processor systems ("2core-pipe-*") with single processor RTOS characteristics and multiprocessor RTOS characteristics. With multiprocessor characteristics, MAE was 0.87%. In contrast, MAE was 3.86% with single processor RTOS characteristics.

There are some amount of errors in estimation results of SystemPerfEst as described above. Nevertheless, it is most important for designers to perform comparative evaluation of mappings in design space exploration. The results in Fig. 11 indicate

that two mappings where one is superior to another over about 10% on FPGA-based prototypes keep their inferior-to-superior relationships on estimation results. In other words, SystemPerfEst cannot differentiate mappings whose difference of performances is in about 10%. However, it may not be crucial problem in a system-level design space exploration since the difference of such mappings with small differences in performance should be discussed with not only performance of them but also their costs such as area and power consumptions.

5.3 Exploration Time

With our integrated framework, we did not changed any of functional description, two profiles of FPGA-based prototypes and could use SystemPerfEst seamlessly.

SystemPerfEst took 0.9 hours for obtaining two input profiles and architecture characteristics using SystemBuilder, and took about 20 seconds for the 22 mappings, 0.9 seconds per a mapping in average. In contrast, FPGA-based evaluation method took about 6.5 hours in total for the 22 mappings, 0.3 hours per a mapping in average. In detail, synthesis by SystemBuilder and a behavioral synthesis tool took about 2.5 hours, logic synthesis and P&R took about 3 hours for nine mappings with dedicated hardware. Software compilation, execution of systems and recording results for 22 mappings took about 1 hour.

With measurement above, estimation time of SystemPerfEst can be formalized as $0.9 + 0.00025 \times N$ hours for N mappings. Evaluation with FPGA-based evaluation method can be formalized as $0.7 \times N$ hours for mappings with some dedicated hardware and $0.05 \times N$ hours for mappings with no dedicated hardware. For the large N, therefore, SystemPerfEst can perform exploration 2,800 times faster than FPGA-based evaluation method for mappings with some dedicated hardware, and 200 times faster even for mappings with no dedicated hardware.

5.4 Effects of Memory Conflicts

In order to examine effects of memory conflicts, we changed mapping of FIFO buffers from on-chip SRAM to off-chip SDRAM, and compared estimation results and evaluation results on FPGA-based prototypes of mappings "2core-pipe-*" and "3core-pipe-*" (shown in **Fig. 12**). In the results, we also showed the comparison of a mapping which all processes are mapped onto a single processor (denoted as "1core" in Fig. 12). The "1core" mapping have no memory conflict in the execution.

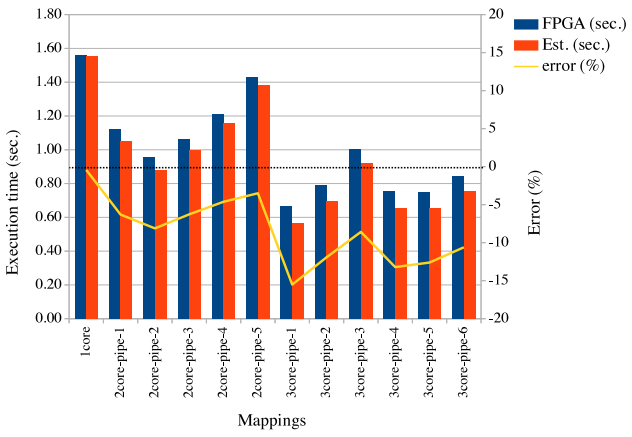


Fig. 12 Performance estimation results of the JPEG decoder system using off-chip SDRAM for FIFO buffers.

First, from results of the "1core" mapping (−0.42% error), we could show that SystemPerfEst can estimate performance of mappings which differ not only processes but also memories accurately.

Then we focused on the results of mappings with two and three processors, errors got worse than those in Fig. 11. MAE increased to 8.47%. Moreover, distribution of errors got wide, from −0.42% to −15.52%. Since processors on target architecture use no cache, the cause of these negative errors is supposed to be memory conflicts on an off-chip SDRAM. If more processors or dedicated hardware are used for more parallelism, estimation errors may get worse than the results in Fig. 12. Therefore techniques which can estimate effects of memory conflicts in short time are necessary, and we are currently working on this topic.

6. Conclusions

In system-level design, system designers describe functionalities as processes and channels, and iterates mapping of processes onto processing elements and evaluation. We proposed a fast performance estimation framework for system-level design exploration, combining our system-level design tool, named SystemBuilder, and a newly developed trace-based simulation tool, named SystemPerfEst.

Since SystemPerfEst works in close cooperation with SystemBuilder, designers easily estimate performance of design candidates exhaustively after describing functionalities of a system. Moreover, with the architecture characterizer description provided by our estimation framework, designers easily reflect characteristics of target FPGAs, memory modules and RTOSs.

We demonstrated the effectiveness of our framework through a case study on design space exploration of a JPEG decoder system. In design space exploration of the JPEG decoder system, performance estimation results of SystemPerfEst achieved 1.92% in mean average error.

Currently we are working for considering the memory conflicts. Also, consideration of caches on processors should be added. Moreover, we are developing an efficient design space exploration strategy and an exploration automation tool which uses SystemPerfEst according to the strategy.

Acknowledgments This work is in part supported by STARC (Semiconductor Technology Academic Research Center).

Reference

[1] Psyco, available from ⟨http://psyco.sourceforge.net/⟩.
[2] TOPPERS Project, available from ⟨http://www.toppers.jp/⟩.
[3] Cai, L., Gerstlauer, A. and Gajski, D.: Retargetable profiling for rapid, early system-level design space exploration, *DAC* (2004).
[4] Dömer, R., Gerstlauer, A., Peng, J., Shin, D., Cai, L., Yu, H., Abdi, S. and Gajski, D.: System-on-chip environment: A SpecC-based framework for heterogeneous MPSoC design, *EURASIP Journal on Embedded Systems*, Vol.2008, pp.1–13 (2008).
[5] Y Explorations, Inc.: eXCite, available from ⟨http://www.yxi.com/index.html⟩.
[6] Gerstlauer, A., Haubelt, C., Pimentel, A., Stefanov, T., Gajski, D. and Teich, J.: Electronic System-Level Synthesis Methodologies, *IEEE*

Trans. Computer-Aided Design of Integrated Circuits and Systems, Vol.28, No.10, pp.1517–1530 (2009).
[7] Ha, S., Kim, S., Lee, C., Yi, Y., Kwon, S. and Joo, Y.-P.: PeaCE: A hardware-software codesign environment for multimedia embedded systems, *ACM Trans. Design Automation of Electronic Systems*, Vol.12, No.3, pp.1–25 (2007).
[8] Honda, S., Tomiyama, H. and Takada, H.: RTOS and Codesign Toolkit for Multiprocessor Systems-on-Chip, *ASP-DAC* (2007).
[9] Kawahara, R., Nakamura, K., Ono, K., Nakada, T. and Sakamoto, Y.: Coarse-grained simulation method for performance evaluation of a shared memory system, *ASP-DAC* (2011).
[10] Keutzer, K., Malik, S., Newton, A.R., Rabaey, J.M. and Sangiovanni-Vincentelli, A.: System Level Design: Orthogonalization of Concerns and Platform-Based Design, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, Vol.19, No.12, pp.1523–1543 (2000).
[11] Mahadevan, S., Virk, K. and Madsen, J.: ARTS: A SystemC-based framework for multiprocessor Systems-on-Chip modelling, *Design Automation for Embedded Systems*, Vol.11, No.4, pp.285–311 (2007).
[12] Pimentel, A.D., Thompson, M., Polstra, S. and Erbas, C.: Calibration of abstract performance models for system-level design space exploration, *Journal of Signal Processing Systems*, Vol.50, No.2, pp.99–114 (2008).
[13] Shibata, S., Ando, Y., Honda, S., Tomiyama, H. and Takada, H.: Efficient Design Space Exploration at System Level with Automatic Profiler Instrumentation, *IPSJ Trans. System LSI Design Methodology*, Vol.3, pp.179–193 (2010).
[14] Thompson, M., Nikolov, H., Stefanov, T., Pimentel, A.D., Erbas, C., Polstra, S. and Deprettere, E.F.: A framework for rapid system-level exploration, synthesis, and programming of multimedia MP-SoCs, *CODES+ISSS* (2007).
[15] Ueda, K., Sakanushi, K., Takeuchi, Y. and Imai, M.: Architecture-level performance estimation method based on system-level profiling, *IEE Proceedings – Computers & Digital Techniques*, Vol.152, No.1, pp.12–19 (2005).
[16] Wild, T., Herkersdorf, A. and Lee, G.-Y.: TAPES – Trace-based architecture performance evaluation with SystemC, *Design Automation for Embedded Systems*, Vol.10, No.2–3, pp.157–179 (2005).

Seiya Shibata received his B.E. degree in Information Engineering and M.S. degree in Information Science from Nagoya University in 2007, and 2009, respectively. Currently he is a Ph.D. candidate at the Graduate School of Information Science, Nagoya University. His research interests include system-level design automation and embedded systems.

Yuki Ando received his B.E. degree in Information Engineering and M.S. degree in Information Science from Nagoya University in 2009, and 2011, respectively. Currently he is a Ph.D. candidate at the Graduate School of Information Science, Nagoya University. His research interests include system-level design automation and embedded systems.

Shinya Honda received his Ph.D. degree in the Department of Electronic and Information Engineering, Toyohashi University of Technology in 2005. From 2004 to 2006, he was a researcher at the Nagoya University Extension Course for Embedded Software Specialists. In 2006, he joined the Center for Embedded Computing Systems, Nagoya University, as an assistant professor, where he is now an associate professor. His research interests include system-level design automation and real-time operating systems. He received the best paper award from IPSJ in 2003. He is a member of IEICE, and JSSST.

Hiroyuki Tomiyama received his Ph.D. degree in Computer Science from Kyushu University in 1999. From 1999 to 2001, he was a visiting postdoctoral researcher with the Center of Embedded Computer Systems, University of California, Irvine. From 2001 to 2003, he was a researcher at the Institute of Systems & Information Technologies/KYUSHU. In 2003, he joined the Graduate School of Information Science, Nagoya University, as an assistant professor, and became an associate professor in 2004. In 2010, he joined the College of Science and Engineering, Ritsumeikan University as a full professor. His research interests include design automation, architectures and compilers for embedded systems and systems-on-chip. He currently serves as an editor-in-chief for IPSJ Transactions on SLDM. He has also served on the organizing and program committees of several premier conferences including ICCAD, ASP-DAC, DATE, CODES+ISSS, and so on. He is a member of ACM, IEEE and IEICE.

Hiroaki Takada is a professor at the Department of Information Engineering, the Graduate School of Information Science, Nagoya University. He is also the executive director of the Center for Embedded Computing Systems (NCES). He received his Ph.D. degree in Information Science from the University of Tokyo in 1996. He was a research associate at the University of Tokyo from 1989 to 1997, and was a lecturer and then an associate professor at Toyohashi University of Technology from 1997 to 2003. His research interests include real-time operating systems, real-time scheduling theory, and embedded system design. He is a member of ACM, IEEE, IEICE, and JSSST.

(Recommended by Associate Editor: *Akihisa Yamada*)