Exploring On-Chip Bus Architectures for Multitask Applications

Sungchan Kim*, Soonhoi Ha**

Abstract—**In this paper we present a static performance estimation technique of on-chip bus architectures. The proposed technique requires the static scheduling of function blocks of a task to analyze bus conflicts caused by simultaneous accesses from processing elements to which function blocks are mapped. To apply it to multitask applications, the concurrent execution of the function blocks of different tasks also should be considered. Since tasks are scheduled independently, considering all cases of concurrency in each processing element is impractical. Therefore we make an average estimate on the effects of other tasks with respect to bus request rate and bus access time. The proposed technique was incorporated with our exploration framework for on-chip bus architectures. Its viability and efficiency are validated by a preliminary example.**

Index Terms—**Performance estimation, multitask application, on-chip bus, design space exploration, queuing theory**

I. INTRODUCTION

Insatiable demand of system performance makes it inevitable to integrate more and more processing elements in a single SoC (System-on-a-Chip) to meet the performance requirement. Using the paradigm of separation between function and architecture and between communication and computation [1], communication architecture decision is made after selecting processing elements and mapping function blocks to them. Separation between computation and communication enables a system designer to explore communication architectures independently of processing element selection and mapping. Since the design space of communication architectures is extremely wide, it is critical to develop a very efficient exploration technique. While diverse interconnection networks are searched for, particularly n the realm of NoC (Network-on-Chip), we are concerned about bus-based communication architectures since it is still mode widely used due to its simplicity and popularity. However, even after a specific bus standard is chosen, the design space of bus architectures can still be huge. For example, we need to determine how many bus segments are used with what topology and which processing elements and memory banks are allocated to which bus segments. We also have to decide memory types and memory system configurations. If we include the selection of bus operation clock frequency and arbitration policy, the design space exploration explodes. Therefore, fast and accurate performance estimation is the key to practical design space exploration methodology.

Furthermore, with the fast evolution of programmable hardware, such as microprocessor or DSP, and ever increasing complexity of its application, many parts of the applications tend to be implemented as software. Such systems let programmable hardware run multiple tasks, which brings up the need for new performance evaluation technique suitable to multitask applications. We propose the performance estimation technique being capable of considering multitask applications, which is the extension of our previous work [9].

In our previous work, an iterative two-step exploration methodology was proposed for bus-based

Manuscript received June 28, 2004; revised December 7, 2004.
School of EECS, Seoul National University, Seoul, Korea
E-mail : *ynwie@iris.snu.ac.kr, **sha@iris.snu.ac.kr

on-chip communication architecture and memory allocation of single-task applications. The proposed method uses static performance estimation technique to reduce a large design space drastically and quickly, and applies trace-driven simulation technique to the reduced set of design candidates for accurate performance estimation.

The remainder of this paper is organized as follows. Section II reviews the related works. Section III provides the brief explanation on our exploration framework. Section IV discusses the proposed estimation technique with a preliminary example. We show some experimental results in Section V and conclude this paper in Section VI.

II. RELATED WORK

For the exploration of communication architectures, simulation-based estimation is widely adopted [3][4]. It gives accurate estimation results but pays too heavy computational cost to be used for exploring the large design space. To overcome this difficulty, a hybrid approach between a static estimation and a simulation approach has been developed by Lahiri, Rahhunathan, and Dey [5]. Since the design space is extremely huge, most previous works focused on a small number of design axes. Gasteier, Munch, and Glensner proposed a bus topology synthesis technique at high level using port constraint of components and cost considering only static information [6]. Meftali, Gharsalli, Fousseau, and Jerraya found a performance-optimal shared memory allocation considering area of communication channels and memory using linear integer programming (ILP) [7]. Lahiri, Rahhunathan, and Dey proposed an exploration technique optimizing component mapping to a bus and bus protocols for a given bus topology [8].

III. REVIEW ON EXPLORATION FLOW

In this section, our framework for exploring on-chip communication architecture is briefly summarized. The inputs of the framework are memory traces and the execution schedule of function blocks in target application. The body of the iteration loop consists of three main steps as shown in Fig. 1.

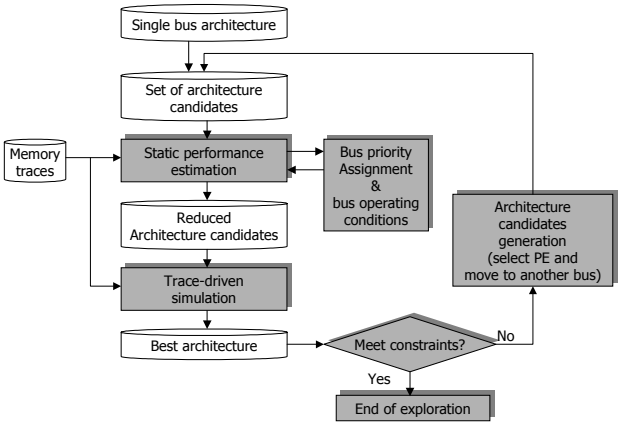


Fig. 1. The design space exploration flow.

In the first step, the design subspace of architecture candidates is explored and is quickly reduced to build a reduced set of design points to be carefully examined. With a given set of architecture candidates, we visit all design points by varying bus operation conditions. Then static performance estimation is applied to select the design points of top 10% in terms of performance because the proposed static estimation method has less than 10 % error around an accurate simulation result [2]. This technique is based on the queuing model of target bus architecture where processing elements are customers and a bus with associated memories is a single server. In order to evaluate the performance of the function block *fb* on a processing element, it requires bus request rate $ar(fb)$, schedule length $sl(fb)$ and average bus access time $ba(fb)$ assuming ideal but unrealistic bus conditions, i.e. no waits for bus grant and the access time of one cycle for unit data transfer from memory. The bus request rate is a ratio of bus access counts $bc(fb)$ over $sl(fb)$. Using those parameters, average delay to obtain the bus grant of each processing element is evaluated as illustrated in Fig. 2 considering bus conflicts. For a task where the schedule of function blocks is pre-determined, the active duration of bus requests are disclosed statically as shown in Fig. 2(a). At the beginning of the schedule, T_0, three function blocks A, B, and C can be executed concurrently on PE_0, PE_1, and PE_2 respectively. To evaluate expected wait delay for bus access from each processing element, the queuing system is constructed as shown in Fig. 2(a). By considering wait for bus grant, bus access time, and internal execution time of each function blocks involved in the queuing system, the earliest completion time of 3 processing

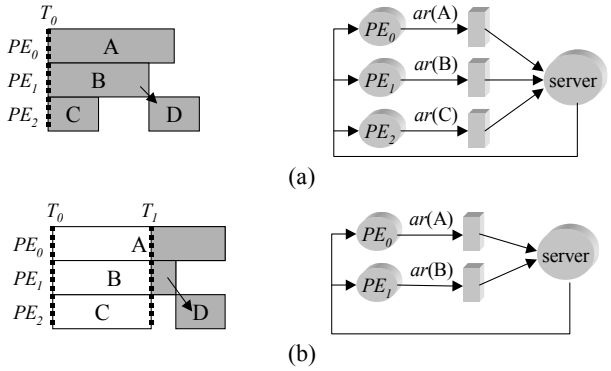


Fig. 2. (a) The example schedule of PE_0, PE_1, and PE_2 and corresponding queuing model and (b) the completion of the function block C at T_1 and the new queuing model after T_1.

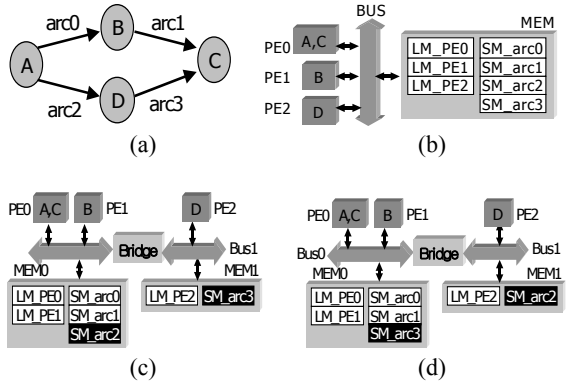


Fig. 3. (a) The behavior specification of an illustrative example, (b) its single-bus implementation, (c) and (d) dual-bus implementations.

elements can be obtained. Note that the schedule length of function blocks is extended by bus grant and access latency that is dependent on the target communication architecture. If the function block C on PE_2 is finished first at T_1, we consider the function blocks that are concurrently executable after that time. Since two function blocks A and B are available as shown in Fig. 2(b), their associated queuing model with PE_0 and PE_1 is constructed. The shaded regions indicate the remains to be evaluated by the static estimation. Such evaluation process is repeated until the queuing model examines all of function blocks.

The second step applies trace-driven simulation to the selected design points from the first step. It accurately evaluates the performance of the design points in the reduced space and determines the best design point. We exit the iteration if given constraints are met at the end of iteration. Otherwise, we go to the third step and repeat another round of iteration.

The third step generates a next set of architecture candidates: from the architecture of the best design point, we explore the design space incrementally by move of a processing element to another existing bus or a new bus and allocation of their associated shared memory segments. For reader's better understanding, an illustrative example is given in Fig. 3. Initially, system behavior is specified as a block diagram of four functions blocks, where arcs represent execution dependency. Fig. 3(b) represents the mapping of function blocks to processing elements and its single-bus implementation. The memory connected to the bus contains seven logical memory segments: three local memory segments and four shared memory segments. The arcs between the function blocks of different processing elements are implemented as shared memory segments for communication between different processing elements. The exploration procedure explained above starts with this single-bus architecture that becomes the only element in the '*set of architecture candidates*' initially. Fig. 3(c) and (d) show two architecture candidates by move of *PE2* to new bus and different allocations of its associated shared memory segments *SM_arc2* and *SM_arc3* from the single bus architecture in Fig. 3(b).

IV. Estimation Of Multitask Applications

This section discusses the extension of our previous performance estimation technique to multitask applications. To clarify the problem, we assume the followings:

- There are no dependency between tasks;
- Any kind of preemptable task scheduling policy can be used;
- The overheads due to scheduling itself are not considered.

First of all, we consider a preliminary example, 4-channel *DVR* (digital video recorder). The *DVR* receives the raw bit streams from external 4 sources and encodes each stream separately using an H.263 encoding algorithm. Each channel corresponds to a task so that

DVR has four tasks, from *ch0* to *ch3*. Fig. 4(a) and (b) show the specification and the schedule of an H.263 encoder respectively. Fig. 4 (c) represents the mapping of function blocks to processing elements in *DVR*. Each of two ARM processors runs one task respectively. Each task is mapped to an ARM processor except for *ME* and *DCT* blocks, which are mapped to the dedicated hardwares, *HE_ME* and *HW_DCT*, respectively. Memory traces are obtained by encoding a P-frame of QCIF-format bit stream.

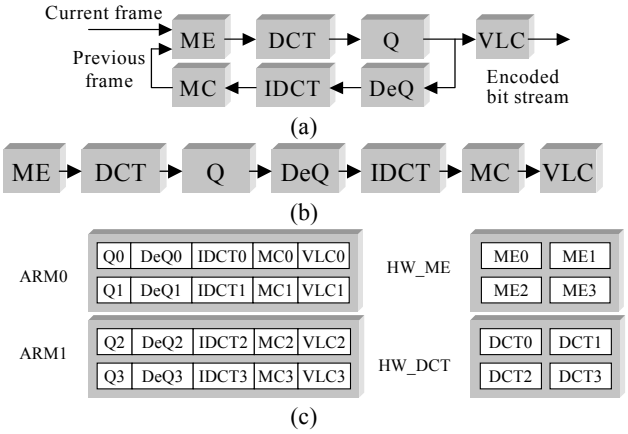


Fig. 4. (a) The specification and (b) the schedule of H.263 encoder and (c) the mapping of function blocks to processing elements for 4-ch *DVR*.

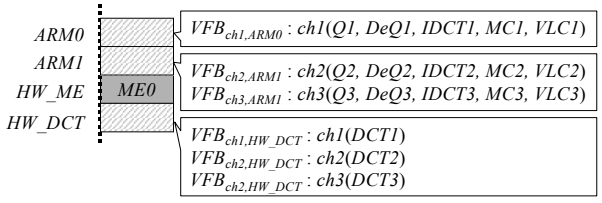


Fig. 5. The function blocks of *ch1*, *ch2*, and *ch3* on *ARM0*, *ARM1*, and *HW_DCT*, which can be executed simultaneously with *ME0*.

In order to evaluate the performance of the task *ch0*, *ME0* is selected first. In *ch0*, no function blocks are concurrently executable with *ME0* due to the execution dependency. On the other hand, the function blocks of other tasks, *ch1*, *ch2*, and *ch3*, can be executed concurrently while *ME0* is executed on *HW_ME*. To build the queuing model as depicted in Fig. 2, the function blocks being concurrently executed on *ARM0*, *ARM1*, and *HW_DCT* with *ME0* should be selected. As shown in Fig. 5, only *ch1* can be simultaneously executable with *ME0* in *ARM0* since *ME0* of the task *ch0* is executed in *HW_ME*. However it can not be statically determined which function block of *ch1* is concurrently executable with *ME0*. Moreover, even in the case of *ARM1*, two tasks *ch2* and *ch3* are available. If we enumerate the queuing models for all possible combinations on concurrency of the function blocks just with *ME0*, $5\times\binom{2}{1}\times5\times3$ or 150 queuing models should be investigated, which is impractical for the fast design space exploration. This difficulty can be solved by an approximated approach. As explained in the previous section, the required parameters for constructing the queuing model are bus request rate and average bus access time. In order to model how a task affects another task approximately, we consider a virtual function block $VFB_{\tau,pe}$ for the task τ on the processing element *pe* with the approximated bus request rate $ar(VFB_{\tau,pe})$ and the bus access time $ba(VFB_{\tau,pe})$ while *ME0* is executed. Thus, $ar(VFB_{ch1,ARM0})$ and $ba(VFB_{ch1,ARM0})$ of $VFB_{ch1,ARM0}$ can be obtained as follows respectively:

$$ar\,(VFB_{\,ch1\,,\,ARM0}\,) = \frac{\sum bc\,(\,fb_{\,ch1\,,\,ARM0}\,)}{sl\,(ch\,1)} \quad (1)$$

and

$$ba(VFB_{ch1,\,ARM0}) = \frac{\sum\{ba(fb_{ch1,\,ARM0})\cdot bc(fb_{ch1,\,ARM0})\}}{\sum bc(fb_{ch1,\,ARM0})}, \quad (2)$$

where $fb_{ch1,ARM0} \in \{$*the function blocks of ch1 mapped to ARM0*$\}$ and $sl(ch1)$ is the schedule length of the task *ch1*. For conservative estimation, we assume the followings: First, in the case of *ARM1* where two virtual function blocks, $VFB_{ch2,ARM1}$ and $VFB_{ch3,ARM1}$, are considered, the virtual function block with higher bus request rate is selected. Second, the schedule length of all virtual function blocks is assumed to be infinite to make them affect *ME0* throughout the entire execution of *ME0*. The construction of the virtual function blocks associated with *ch0* is required on every completion of the function blocks being estimated in *ch0*.

In general, we define two terms $T(fb)$ and $P(fb)$, which are the task that has the function block *fb* and the processing element executing the function block *fb*

respectively. If two function blocks *fb* and *fb** are to be executed concurrently according to the static task schedule, we denote it by *fb*//*fb**. Suppose that the function block *fb** is executed on the processing element *pe**, i.e. $P(fb^*)=pe^*$. In order to build the queuing system of Fig. 2, $\psi_{fb^*,pe}$ is defined as the virtual function block of the processing element *pe*, which is seen being executed concurrently with the function block *fb**. Therefore, the bus request rate $ar(\psi_{fb^*,pe})$ of $\psi_{fb^*,pe}$ becomes

$$ar(\psi_{fb^*,pe}) = \max_{\forall \tau,\ \tau \neq \tau^*} \left\{ \frac{\sum_{fb_i} bc(fb_i)}{sl(\tau)} \right\}, \qquad (3)$$

where $fb_i \in \{ fb \mid T(fb)=\tau,\ fb//fb^*,\ P(fb)=pe,\ pe \neq pe^* \}$ and $\tau^*=T(fb^*)$. If the task τ is selected to build the virtual function block of *pe* by (3), the average bus access time $ba(\psi_{fb^*,pe})$ of the virtual function block $\psi_{fb^*,pe}$ with no wait for bus grant is

$$ba(\psi_{fb^*,pe}) = \frac{\sum_{fb_i} \{ba(fb_i) \cdot bc(fb_i)\}}{\sum_{fb_i} bc(fb_i)}, \qquad (4)$$

where $fb_i \in \{ fb \mid T(fb)=\tau,\ P(fb)=pe,\ pe \neq pe^* \}$. With those parameters of each processing except the processing element *pe** using (3) and (4), the queuing system of *pe** and others can be constructed to get the average wait for bus grant of the function block *fb**.

Note that we can not consider the preemption of a task by others, which occurs usually in the real-time scheduling using the fixed or dynamic priorities assigned to tasks [10][11]. This is because that the proposed technique is focused on the case when tasks are put in execution. If we extract the maximum execution time of tasks as constraints through well-known real-time schedulability analysis techniques [10][11], the execution time of tasks can be translated into bus access time in terms of clock cycles for each processing element. Then, the proposed technique is used for finding out the bus architecture that satisfies all constraints of processing elements.

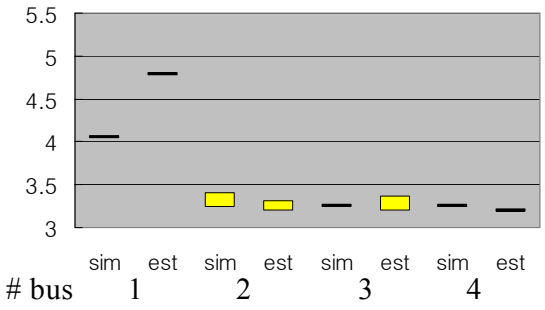
V. EXPERIMENTS

In this section, we validate our proposed estimation technique with the *DVR* example. The experiments are about the comparison of execution time in terms of average bus access time using both our proposed technique and a trace-driven simulation. To perform the trace-driven simulation, we use in-house simulator that adopts a simple bus protocol, where the advanced features such as address/data bus pipelining, split-transaction, multiple-outstanding masters, and so on are not modeled. The trace-driven simulator schedules tasks using the rate-monotonic scheduling with fixed priorities [10]. However no overheads that occur on scheduling tasks are considered in the simulator. All of experiments were conducted on Xeon 2.8GHz workstation running Linux.

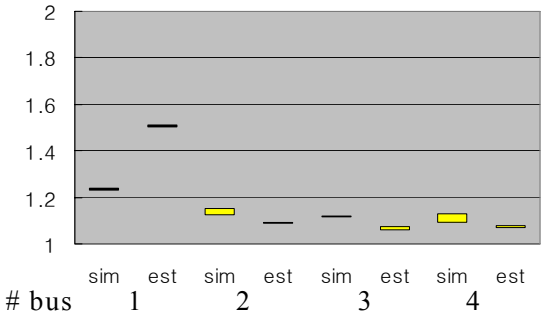
Table 1 represents the results of exploration for *DVR*. The exploration is maintained until no more performance gain is obtained. In Table 1, the second column '*number of arch*' shows the number of generated architecture candidates according to the associated number of buses during the exploration. The number of processing elements becomes the maximum number of buses that an architectures candidate can have. The second row from the bottom is the average elapse of estimating an architecture candidate in the first exploration step. It takes no less than one second, which tells us the effectiveness of the proposed technique. In the third column '*Execution time*', the best performance according to the number of buses using the trace-driven simulation is recorded in terms of bus clock cycles. Performance gain is no longer obtained with more than 3 buses, where the overhead of crossing bus bridges tends to exceed the benefit of scattering bus traffic by splitting a bus.

Table 1. The results of exploration for 4-ch *DVR*.

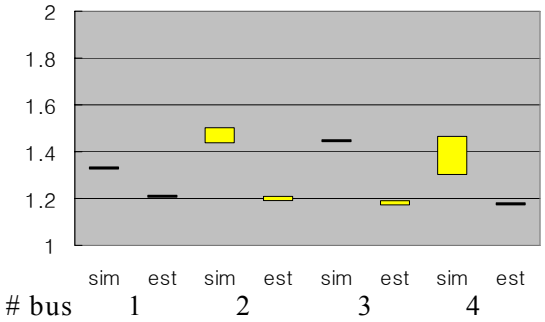
Number of buses	Number of arch	Execution time
1	1	24869021
2	2208	22297455
3	1072	22502381
4	512	22486609
Total	3793	
Estimation/arch in 1st step (sec)	0.83	
CPU time (sec)	3918	



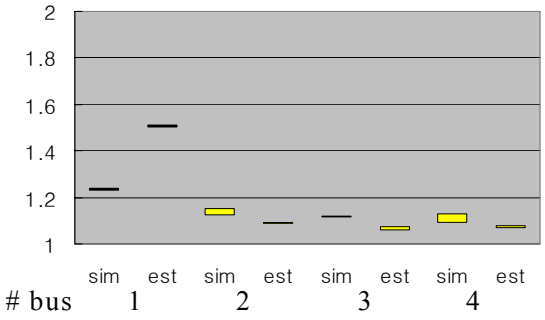
(a) *ARM0*



(b) *ARM1*



(c) *HW_ME*



(d) *HW_DCT*

Fig. 6. The estimated bus access time compared with those by simulation for each processing element in *DVR*.

Fig. 6 shows the average bus access time of unit data from each processing element on *DVR* by both proposed static estimation and trace-driven simulation according to the number of buses. The estimation error compared with the trace-driven simulation is about 28% in the worst case. Although it appears to be large, the ratio of bus access time over the entire execution does not exceed 30%. Thus the amount how much the error affects on the entire execution is less than 10 %. Furthermore, average estimation error is just about 6 %. Through the experiments, we verify that the proposed estimation technique can be used successfully for exploration of multitask applications.

VI. Conclusions

In this paper we proposed the static performance estimation technique of on-chip bus architectures for multitask applications. The proposed technique is based on the queuing model of processing elements associated with a set of function blocks that can be executed concurrently. To construct a set of function blocks considering concurrency between tasks, a virtual function block corresponding to a task in each processing element is introduced with approximated parameters for the queuing analysis. Experimental results show that the proposed technique is acceptable for the fast design space exploration of communication architecture with reasonable estimation error.

Acknowledgments

This work was supported by National Research Laboratory Program (number M1-0104-00-0015), Brain Korea 21 Project, and IT-SoC project. ICT at Seoul National University provided research facilities for this study.

References

[1] K. Keutzer, S. Malik, R. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli, “System-level design: Orthogonalization of concerns and platform-based design,” *IEEE Trans. Computer-Aided Design*, vol.19, pp. 1523-1543, Dec, 2000.

[2] S. Kim, C. Im, and S. Ha, "Schedule-aware performance estimation of communication architecture for efficient design space exploration," in *Proc. Intl. Conf. Hardware/Software Codesign and System Synthesis*, pp. 195-200, Oct. 2003.

[3] P. Lieverse, P. van der Wolf, E. Deprettere, and K. Vissers, "A methodology for architecture exploration of heterogeneous signal processing systems," in *Proc. IEEE Workshop Signal Processing Systems*, pp. 181-190, Oct, 1999.

[4] J. A. Rowson and A. Sangiovanni-Vincentelli, "Interface based design," in *Proc. Intl. Conf. Design Automation*, pp. 178-183, Jun, 1997.

[5] K. Lahiri, A. Raghunathan, and S. Dey, "System-level performance analysis for designing system-on-chip communication architecture," *IEEE Trans. Computer-Aided Design*, vol.20, pp. 768-783, Jun, 2001.

[6] M. Gasteier, M. Munch, and M. Glensner, "Generation of interconnect topologies for communication synthesis," in *Proc. Intl. Conf. Design Automation and Test in Europe*, pp. 36-43, Feb, 1998.

[7] S. Meftali, F. Gharsalli, F. Rousseau, and A. A. Jerraya, "An optimal memory allocation for application-specific multiprocessor system-on-chip," in *Proc. Intl. Symp. System Synthesis*, pp. 19-24, Oct, 2001.

[8] K. Lahiri, A. Raghunathan, and S. Dey, "Efficient exploration of the SoC communication architecture design space," in *Proc. Intl. Conf. Computer Aided Design*, pp. 424-430, Nov, 2000.

[9] S. Kim, C. Im, and S. Ha, "Efficient exploration of on-chip communication architecture and memory allocation," in *Proc. Intl. Conf. Hardware/Software Codesign and System Synthesis*, pp. 248-253, Sep. 2004.

[10] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *Journal of ACM*, vol.20, pp. 26-61, Jan. 1973.

[11] N. Audsley, A. Burns, M. Richardson, and A. Wellings, "Hard real-time scheduling: The deadline-monotonic approach," In *Proc. IEEE Workshop Real-Time Operating Systems and Software*, pp. 133-137, May 1991.

Sungchan Kim received the B.S. degrees in material science and engineering and the M.S. degrees in computer engineering from Seoul National University, Seoul, Korea, in 1998 and 2000, respectively. He is currently working toward the Ph.D. degrees at the same university.

His research interests include Hardware/Software codesign, analysis of performance and power consumption, and architecture optimization of SoC for multimedia applications.

Soonhoi Ha received the B.S. and M.S. degrees in electronics engineering from Seoul National University, Seoul, Korea, in 1985 and 1987 respectively, and the Ph.D. in electrical engineering and computer science from University of California, Berkeley, CA, in 1992.

He worked for Hyundai Electronics Industries Corporation from 1993 to 1994 before he joined the faculty of the school of electrical engineering and computer science at Seoul National University. He is currently an associated professor. His primary research interests are various aspects of embedded system design including Hardware/Software codesign and design methodologies.