Schedule-Aware Performance Estimation of Communication Architecture for Efficient Design Space Exploration

Sungchan Kim, Chaeseok Im, and Soonhoi Ha, *Member, IEEE*

***Abstract*—In this paper, we are concerned about performance estimation of bus-based communication architectures assuming that task partitioning and scheduling on processing elements are already determined. Since communication overhead is dynamic and unpredictable due to bus contention, a simulation-based approach seems inevitable for accurate performance estimation. However, it is too time-consuming to be used for exploring the wide design space of bus architectures. We propose a static performance-estimation technique based on a queueing analysis assuming that the memory traces and the task schedule information are given. We use this static estimation technique as the first step in our design space exploration framework to prune the design space drastically before applying a simulation-based approach to the reduced design space. Experimental results show that the proposed technique is several orders of magnitude faster than a trace-driven simulation while keeping the estimation error within 10% consistently in various communication architecture configurations.**

***Index Terms*—Communication architecture, design space exploration, performance estimation, queueing theory.**

I. Introduction

INSTIABLE demand of system performance makes it inevitable to integrate more and more processing elements in a single system-on-a-chip (SoC) to meet the performance requirements. As a new design paradigm for such high-performance SoCs, the separation between function and architecture and between communication and computation is recently proposed [1], [23]. Adapting this paradigm, it is assumed that system behavior is modeled as a composition of function blocks, and communication architecture is determined after a decision is made on which processing elements are used and which function blocks are mapped on which processing elements. Therefore, it allows designers to explore communication architectures independently of component selection and mapping.

While diverse interconnection networks are searched for, particularly in the realm of network-on-chip (NoC) design, we are concerned about bus-based communication architectures since they are still the most widely used due to their 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 topologies 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 explodes.

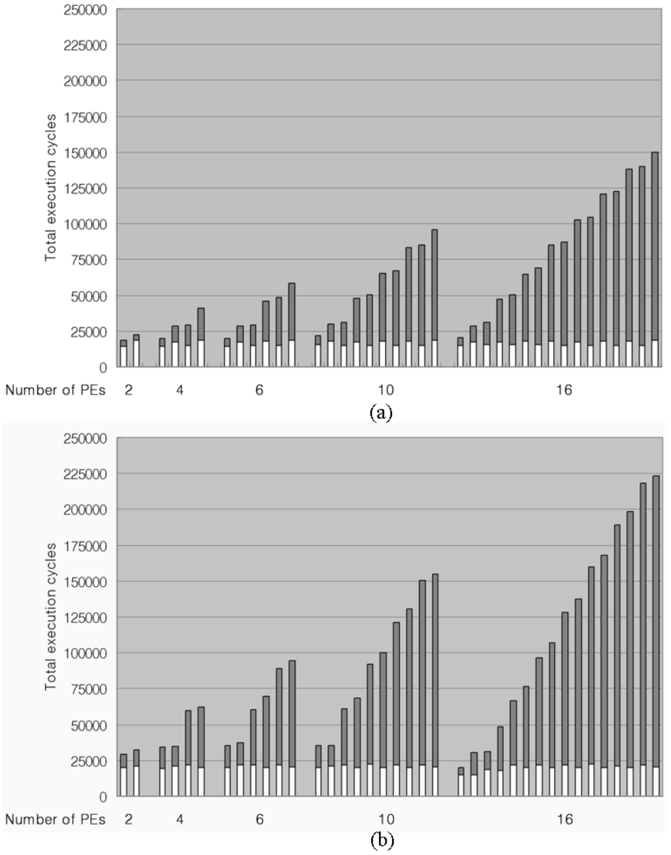


Fig. 1. Performance variation due to dynamic bus conflicts on a single bus for two bus request rates: (a) 0.115 (typical for multimedia applications) and (b) 0.17 (highly intensive cases).

Since a bus is a shared medium between multiple processing elements that compete with each other for using it, communication overhead is highly unpredictable due to bus contention. Fig. 1 shows the performance variation due to such dynamic conflicts in terms of bus clock cycles on the single bus complying with the AMBA AHB specification [21] varying the number of processing elements. Each processing element

Manuscript received March 13, 2004; revised June 4, 2004, and October 16, 2004. This work was supported by the National Research Laboratory under Program M1-0104-00-0015, Brain Korea 21 Project, and the IT-SoC project. ICT at Seoul National University provided research facilities for this study.

S. Kim and S. Ha are with the Department of Electrical Engineering and Computer Science, Seoul National University, Seoul 151-742, Korea (e-mail: ynwie@iris.snu.ac.kr; sha@iris.snu.ac.kr).

C. Im is with Samsung Advanced Institute of Technology, Yongin, Gyeonggi 440-600, Korea (e-mail: csim@iris.snu.ac.kr).

Digital Object Identifier 10.1109/TVLSI.2004.842912

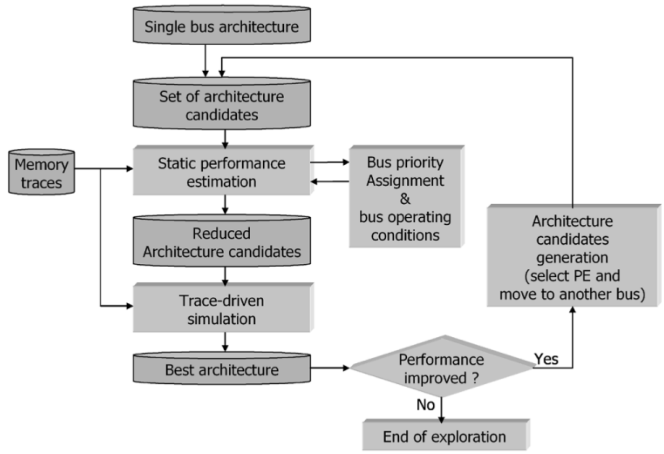

Single bus architecture
Set of architecture candidates
Memory traces
Static performance estimation
Bus priority Assignment & bus operating conditions
Reduced Architecture candidates
Architecture candidates generation (select PE and move to another bus)
Trace-driven simulation
Best architecture
Performance improved ?
Yes
No
End of exploration

Fig. 2. Proposed design space exploration flow.

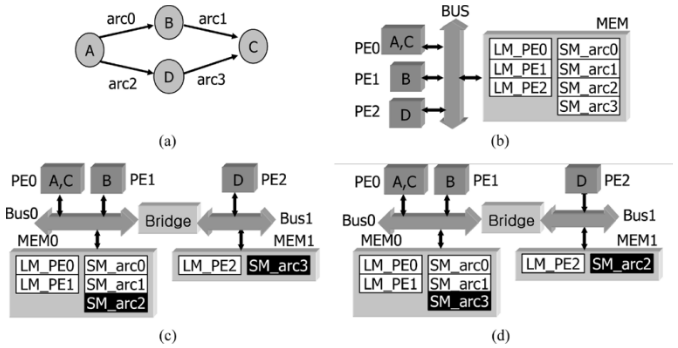


Fig. 3. (a) Behavior specification of an illustrative example. (b) Its single-bus implementation. (c), (d) Dual-bus implementations, which are different in the mapping of shared memories *SM_arc2* and *SM_arc3*.

issues bus requests randomly with a given request rate. The request rate of 0.1 means that 10% of the total execution time is used for bus accesses. The average request rate of processing elements in Fig. 1(a) is 11.5%, which is a typical bus request rate in multimedia applications such as H.263 encoder/decoder, MP3 decoder, and so on. Fig. 1(b) shows the case of and average request rate of 17%, which can be thought of as intensive bus traffic. The execution time of each processing element is divided into two sections: white and dark. The dark section indicates the waiting time for the bus grant while the white section shows the actual execution time including the bus access time. The figure shows that the overhead due to bus conflicts becomes the dominating factor on the entire execution time as the number of processing elements increases and the bus request rate becomes higher. Thus, it is critical to consider the bus conflicts for accurate performance estimation.

In order to explore the wide design space, we need to estimate the performance of communication architectures fast as well as accurately. While a simulation-based approach is widely used for accurate estimation, it is too slow to explore the huge design space. To overcome this drawback, our communication architecture exploration framework consists of two techniques: static performance estimation and trace-driven simulation. We focus on the static performance estimation technique, which is used to prune the design space drastically before the trace-driven simulation technique is applied. As the static estimation gets more accurate, the design space becomes smaller. The proposed static estimation technique computes the expected waiting time due to bus contention through the queueing analysis.

The remainder of this paper is organized as follows. In Section II, we briefly present our design space exploration framework. Section III reviews some related works and summarizes our contributions. In Sections IV–VI, the static estimation methods based on the queueing models for fixed priority, round-robin, and two-level time-division multiple-access (TDMA) arbitration based bus systems are explained, respectively. Section VII contains the detailed discussions on the estimation method considering the schedule information and the extension to multiple bus systems. The effect of schedule complexity on the estimation time is discussed in Section VIII. In Section IX, we provide the experimental results and draw conclusions in Section X.

II. Proposed Design Space Exploration Framework

The proposed design space exploration flow is shown in Fig. 2. We assume that the task schedule on each processing element and the memory trace information, including both local and shared memory accesses, from each processing element are given. The memory traces are obtained using a cycle-accurate instruction set simulator for each processor core and an HDL simulator for ASIC parts after the mapping is completed.

We traverse the design space in an iterative fashion, starting with a single bus architecture that becomes the only element in the "*set of architecture candidate*" initially and the best architectures at the end of each iteration. From the best architecture of the previous iteration, we explore the design space incrementally by selecting a processing element and allocating it to a different bus or a new bus. For a better understanding of the exploration flow and the target architectures of interest, we use an illustrative example of Fig. 3. The system behavior is specified as the block diagram of four functions blocks. The arcs between the function blocks show the execution dependency. Those function blocks are mapped to three processing elements: A and C are mapped to the processing element *PE0*, B to *PE1*, and D to *PE2*, respectively.

Fig. 3(b) represents a single bus implementation. Its memory subsystem contains seven logical memory segments: three local memory segments and four shared memory segments. The memory segments *LM_PE0*, *LM_PE1*, and *LM_PE2* are the local memory segments of *PE0*, *PE1*, and *PE2*, respectively. The arcs between function blocks are implemented as shared memory segments for intercomponent communication.

In the first iteration, the single bus architecture in Fig. 3(b) becomes the best architecture. Now we go into the second iteration and generate the candidate architectures by selecting a processing element and allocating it to a different bus. Suppose that we select *PE2* and allocate it to a new bus resulting in a dual bus system. Since all local memory segments should reside in the same bus as the associated processing elements, there are four candidate architectures depending on where to put the shared memory segments associated with *PE2: SM_arc2* and *SM_arc3*. The function blocks A and D use the shared memory segment *SM_arc2* so that it may be allocated to either *bus0* or *bus1*. However, *SM_arc0* that is accessed by A and B should remain at *bus0*. Among four candidate architectures, Fig. 3(c) and (d) shows two candidate architectures. In the case that we select *PE0* and move it to a new bus, we generate 16 different candidate architectures since *PE0* is associated with four shared memory segments. In this fashion, we can generate 24 candidate architectures in the second round of iteration by moving a processing element into a new bus and considering all possible shared memory segment allocations. In addition, we consider the different priority assignments of processing elements on each bus.

The proposed static estimation technique is applied to all candidate architectures to select the reduced set of candidate architectures. Suppose that the accuracy of the static estimation is 10%. Then we select the top 10% of the candidate architectures in terms of the estimated performance. Among the reduced set of candidate architectures, we select the best candidate architecture through the accurate trace-driven simulation. Then, we go to the next iteration. A more detailed description of the framework is beyond the focus of this paper and is given in [20].

III. Related Work and Our Contribution

Some researchers have considered communication architecture selection simultaneously during the synthesis of the computation parts of a system and the mapping step. Since the communication overhead is needed for the mapping decision, the static estimation of the communication architecture has been investigated. Knudsen and Madsen estimated the communication overhead taking into account the data transfer rate variation depending on protocol, configuration, and different operating clock frequencies of components [6], [7]. A technique has been proposed to estimate the communication delay using the worst-case response analysis of the real-time scheduling [8]. Ortega and Borriello took into account the static information such as data transfer size, bus protocol overhead, and bus bandwidth to estimate the worst-case bus delay [9]. Nandi and Marculescu proposed the performance measure technique based on a continuous-time Markov process [10]. Daveau *et al.* considered only static information, such as maximum bandwidth of channel,\ and average and peak bandwidth of a processing element, to estimate the performance of communication links between processing elements [11]. Drinic *et al.* used the profiled statistics of communication traffics between cores for a given application for core-to-bus assignment [22]. Thepayasuwan and Doboli proposed the bus architecture synthesis technique that minimizes the cost considering bus topology, communication conflict, and bus utilization using a simulated annealing [25]. However, these techniques do not model the dynamic effects such as bus contention and explore only the limited configuration space.

For the exploration of communication architectures, a simulation-based estimation has been widely adopted in many academic research projects [3]–[5], [24] and commercial tools at various abstraction levels, at the transaction level [13], [14], or at the pin-level [15]. The simulation-based method gives accurate estimation results but pays too heavy a computational cost to be used for exploring the large design space. Thus, the research based on this method cannot only exploit a few design axes to reduce the design space.

Lahiri *et al.* presented the hybrid approach combining static estimation and simulation [12]. In their work, communication and computation segments are grouped to make a bus and synchronization event (BSE) graph from the trace data obtained after system cosimulation. They focused on intercomponent communication activities that are usually localized in time at the boundary of computation segment. The trace groups are scheduled on a communication media, which are shifted by the estimated delays considering the resource contention. They use some static analysis to group the traces and apply a trace-driven simulation with the trace groups. Their approach is similar to ours in that they apply some static analysis to the traces to reduce the time complexity of the trace-driven simulation. However, their approach converges to the trace-driven simulation as the memory traces become larger since the BSE graph size is dependent on the memory traces. On the other hand, our proposed technique extracts only the statistical parameters from the traces. Therefore, the run time of our technique is independent of the trace size.

Our work is inspired by Brandwajn's work [2] in which a simple queueing model of an SCSI bus is proposed, which is summarized in Section IV-A. The model produces remarkable results compared with the simulation results. Since the communication behavior of a processor bus is quite different from that of an I/O bus, however, their approach cannot be directly applicable. Instead, we make several extensions to improve the estimation accuracy significantly. First, the queueing model itself is modified for the processor bus system. Second, based on the fixed priority model, we develop a novel way of modeling other types of buses, such as round-robin and two-level TDMA buses. The extended models are explained in Sections IV-B, V, and VI, respectively. Third, we make use of the task schedule information by the aid of a system-level specification to consider the burstness of bus requests. Finally, the model of a single bus is extended to a multiple bus system, which makes the proposed estimation technique viable for communication architecture exploration considering various bus topologies. The details are given in Section VII.

The key contributions of this paper can be summarized as follows. First, we propose the efficient static-estimation technique using the queueing model to take into account dynamic behaviors due to bus contention and dynamic memory traces. Second, by adopting the task schedule information, we make the accuracy of the estimated entire execution time comparable with that of the trace-driven simulation. Finally, we enable designers to explore the wide design space of communication architectures considering the design axes than the previous works.

IV. Queueing Model of a Fixed Priority Based Bus

A. Base Queueing Model for a Single I/O Bus

A base estimation technique using a queueing model for a fixed priority based I/O bus is reviewed in this section. The basic idea and the notations used here are borrowed from [2]. There are N processing elements $(PE_0, PE_1, \ldots, PE_{N-1})$ competing for the use of a bus. It is assumed that a bus arbitration is based on the fixed priorities of processing elements. PE_0 is assigned the highest priority. The bus access is assumed to be nonpreemptive.

Fig. 4 shows the queueing model of a single bus architecture. λ_i denotes the rate at which the processing element PE_i issues memory requests. It is computed as the ratio between the memory access counts and the scheduled length of execution. If the execution time is lengthened due to bus contention, the effective arrival rate of requests becomes smaller than λ_i. We denote

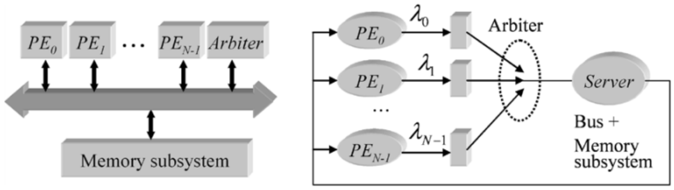


Fig. 4. Queueing model of a single bus.

the actual memory access rate by θ_i, which is actually seen on the bus. The mean service rate of a server for the request from PE_i is denoted by μ_i and its mean service time is the reciprocal of the service rate, i.e., $1/\mu_i$. Let k_i be the expected number of requests from PE_i waiting for use of the bus. It is within the range of $[0, 1]$ if PE_i does not issue the next memory request until the current request is served. Also, we denote w_i as the expected waiting time of the stalled request. Then, we obtain the following equation:

$$\theta_i = (1 - k_i - u_i)\lambda_i \tag{1}$$

where $u_i = \theta_i/\mu_i$ is the bus utilization factor of PE_i. Little's law [16] says

$$w_i = \frac{k_i}{\theta_i}. \tag{2}$$

We want to obtain w_i from (2), which indicates the delays incurred from bus contention. We can extract λ_i from the memory traces. By the memory system and the average burst length of the memory traces, μ_i is determined statically. There remains an unknown parameter k_i in the right side of (1). To obtain this, we use a state transition diagram and its steady-state probability.

As the simple model of a single bus with N processing elements, each processing element has one of three states: computation (no bus request), waiting for bus grant, and bus access, and a bus lies in one of $N + 1$ states depending on which processing element is currently using the bus, including the idle state. These N processing elements and the bus compose the state space of the bus system by $N + 1$ tuples. Although this model is accurate, the number of states explodes exponentially with the increase of N. For example, if N equals 10, a system has $3^{10} \times 11$ or 590 490 states. This means that this simple model cannot be used for fast architecture exploration. Thus, we use an approximate but effective state transition diagram while preserving the accuracy within a certain limit.

We draw a state transition diagram from the viewpoint of each $PE_i (i = 1, \ldots, N - 2)$. We define the system state as a quadruple and its steady-state probability by $p(n_i, n_h, n_l, s)$, where there are $n_i (n_i = 0, 1)$ requests of PE_i, $n_h (n_h = 0, \ldots, i - 1)$ requests of the processing elements with a higher priority than PE_i, and $n_l (n_l = i + 1, \ldots, N - 1)$ requests of the processing elements with a lower priority than PE_i and s ($s =$ 'i', 'h', 'l', '$idle$') is the priority group that uses the bus currently. We also define a set of processing elements with a higher priority as Φ_h and a set of processing elements with a lower priority as Φ_l, respectively. The total numbers of processing elements in Φ_h and Φ_l are defined by N_h and N_l, which equal i and $N - i - 1$, respectively.

The state (n_i, n_h, n_l, s) moves to the state $(n_i, n_h + 1, n_l, s)$ if another request arrives from a processing element in Φ_h and its transition rate is approximated by α_h, $\alpha_h \approx (N_h - n_h)\gamma_h$, where γ_h is the average rate of arrivals for the requests from Φ_h. γ_h can be computed by summing up (1) from PE_0 to PE_{i-1} as

$$\sum_{j=0}^{i-1} \theta_j = \left\{ N_h - \sum_{j=0}^{i-1} (k_j + u_j) \right\} \cdot \gamma_h. \tag{3}$$

The transition rate β_h to the state $(n_i, n_h - 1, n_l, s)$ is approximated as follows:

$$\beta_h \approx \sum_{j=0}^{i-1} \left(\frac{\mu_j \cdot u_j}{\sum\limits_{k=0}^{i-1} u_k} \right). \tag{4}$$

Similar equations can be given to the requests from Φ_l. The readers are referred to [2] for more details on the derivations of α_h, β_h, α_l, and β_l, where α_l and β_l are the average request rate and the mean service time of Φ_l, respectively. From the state transition diagram and the transition rates, we can compute the steady-state probability $p(n_i, n_h, n_l, s)$ using the following additional requirement:

$$\sum_{n_i=0}^{1} \sum_{n_h=0}^{N_h} \sum_{n_l=0}^{N_l} \sum_{s \in \{'i','h','l','idle'\}} p(n_i, n_h, n_l, s) = 1.$$

After all steady-state probabilities are computed, we can compute the expected number of waiting requests k_i by summing up the probabilities of a certain set of states as follows:

$$\sum_{n_h=0}^{N_h} \sum_{n_l=0}^{N_l} \sum_{s \in \{'i','h','l','idle'\}} p(1, n_h, n_l, s) = u_i + k_i. \tag{5}$$

Note that the evaluation of $p(n_i, n_h, n_l, s)$ needs k_i by (3) and (5). Therefore, these equations should be solved by an iterative method until k_i becomes stable. Also, this iterative procedure to get k_i is repeated for each priority level i ($i = 1, \ldots, N - 2$). We used a linear program package to solve the equations and found that the solutions are converged within less than ten iterations in all cases, which takes much less than the trace-driven simulation.

B. Extension to Processor Bus

The base queueing model of the previous section assumes a continuous system where the bus request can be served at any instance of time. In reality, however, bus arbitration is performed at discrete sampling points (i.e., clock edges) among all bus requests accumulated so far. If we assume that there occurs only one event, either request arrival or service completion, in each clock period, the base queueing model may be used as an approximate model. This assumption is suitable for the I/O bus case where bus requests are infrequent compared with the service time and the service time is relatively large. However, several events are very likely to occur during a single clock period in our case. Thus, we modify the base queueing model to allow simultaneous events. The state transition rate of a transition arc should be replaced by the state transition probability within a clock period.

As explained in the previous section, the state of the entire bus system is approximated by a set of state transition diagrams drawn for each processing element. To draw each state transition diagram, we divide the processing elements into three groups:

TABLE I

STATE TRANSITIONS FROM THE STATE (N_I, N_H, N_L, S) TO OTHER STATES

Type 1			Type 2		
The next state	The value of s in the current state	The transition rate	The next state	The value of s in the current state	The transition rate
$(n_i+1,n_h,n_l,\mathrm{s})$	*idle*	$\lambda_i\cdot(1-d_h)\cdot(1-d_l)$	$(n_i+1,n_h+1,n_l,\mathrm{s})$	*idle*	$\lambda_i\cdot d_h\cdot(1-d_l)$
	h	$\lambda_i\cdot(1-d_h)\cdot(1-d_l)\cdot(1-\beta_h)$		*h*	$\lambda_i\cdot d_h\cdot(1-d_l)\cdot(1-\beta_h)$
	l	$\lambda_i\cdot(1-d_h)\cdot(1-d_l)\cdot(1-\beta_l)$		*l*	$\lambda_i\cdot d_h\cdot(1-d_l)\cdot(1-\beta_l)$
$(n_i,n_h+1,n_l,\mathrm{s})$	*idle*	$(1-\lambda_i)\cdot d_h\cdot(1-d_l)$	$(n_i,n_h+1,n_l+1,\mathrm{s})$	*idle*	$(1-\lambda_i)\cdot d_h\cdot d_l$
	i	$d_h\cdot(1-d_l)\cdot(1-\mu_i)$		*i*	$d_h\cdot d_l\cdot(1-\mu_i)$
	h	$(1-\lambda_i)\cdot d_h\cdot(1-d_l)\cdot(1-\beta_h)$		*h*	$(1-\lambda_i)\cdot d_h\cdot d_l\cdot(1-\beta_h)$
	l	$(1-\lambda_i)\cdot d_h\cdot(1-d_l)\cdot(1-\beta_l)$		*l*	$(1-\lambda_i)\cdot d_h\cdot d_l\cdot(1-\beta_l)$
$(n_i,n_h,n_l+1,\mathrm{s})$	*idle*	$(1-\lambda_i)\cdot(1-d_h)\cdot d_l$	$(n_i+1,n_h,n_l+1,\mathrm{s})$	*idle*	$\lambda_i\cdot(1-d_h)\cdot d_l$
	i	$(1-d_h)\cdot d_l\cdot(1-\mu_i)$		*h*	$\lambda_i\cdot(1-d_h)\cdot d_l\cdot(1-\beta_h)$
	h	$(1-\lambda_i)\cdot(1-d_h)\cdot d_l\cdot(1-\beta_h)$		*l*	$\lambda_i\cdot(1-d_h)\cdot d_l\cdot(1-\beta_l)$
	l	$(1-\lambda_i)\cdot(1-d_h)\cdot d_l\cdot(1-\beta_l)$	$(n_i-1,n_h+1,n_l,\mathrm{s})$	*i*	$d_h\cdot(1-d_l)\cdot\mu_i$
$(n_i-1,n_h,n_l,\mathrm{s})$	*i*	$(1-d_h)\cdot(1-d_l)\cdot\mu_i$	$(n_i-1,n_h,n_l+1,\mathrm{s})$	*i*	$(1-d_h)\cdot d_l\cdot\mu_i$
$(n_i,n_h-1,n_l,\mathrm{s})$	*h*	$(1-\lambda_i)\cdot(1-d_l)\cdot\beta_h$	$(n_i+1,n_h-1,n_l,\mathrm{s})$	*h*	$\lambda_l\cdot(1-d_l)\cdot\beta_h$
$(n_i,n_h,n_l-1,\mathrm{s})$	*l*	$(1-\lambda_i)\cdot(1-d_h)\cdot\beta_l$	$(n_i,n_h-1,n_l+1,\mathrm{s})$	*h*	$(1-\lambda_i)\cdot d_l\cdot\beta_h$
			$(n_i+1,n_h,n_l-1,\mathrm{s})$	*l*	$\lambda_i\cdot(1-d_h)\cdot\beta_l$
			$(n_i,n_h+1,n_l-1,\mathrm{s})$	*l*	$(1-\lambda_i)\cdot d_h\cdot\beta_l$

the processing element of interest, those with a higher priority, and those with a low priority. In this approximate model, an event is defined by a single increment or decrement for each coordinate of the state (n_i, n_h, n_l, s). There are two kinds of events which are the bus requests from three priority groups and the completion of current bus access. Table I shows all possible state transitions in our compromised model. We classify the transitions into two types, *Type 1* and *Type 2*, depending on how many events occur during the next clock cycle. Note that we do not allow the state transition from (n_i, n_h, n_l, s) to $(n_i, n_h + 2, n_l, s)$ in the compromised model.

For the computation of the transition probabilities, we define two parameters d_h and d_l as the probabilities that at least one processing element issues a new request from Φ_h and Φ_l, respectively

$$d_h = 1 - (1 - \gamma_h)^{N_h - n_h}, \quad d_l = 1 - (1 - \gamma_l)^{N_l - n_l}. \tag{6}$$

In the case that a processing element in Φ_h is using a bus, the transition probability from (n_i, n_h, n_l, s) to $(n_i, n_h + 1, n_l + 1, s)$ becomes $(1 - \lambda_i) \cdot d_h \cdot d_l \cdot (1 - \gamma_h)$.

In order to validate the accuracy of our compromised model, we compare it with a more general but complicated model to deal with K simultaneous events. In this model, we also allow a certain priority group to request more than one simultaneous event. Then the transition from the state (n_i, n_h, n_l, s) to the state $(n_i^*, n_h^*, n_l^*, s^*)$ is valid as long as the following constraints are satisfied:

$$\begin{aligned} &0 \le n_i^* \le 1;\ n_h - 1 \le n_h^* \le N_h;\ n_l - 1 \le n_l^* \le N_l, \\ &|n_i - n_i^*| + |n_h - n_h^*| + |n_l - n_l^*| \le K. \end{aligned} \tag{7}$$

The rate of the transition from (n_i, n_h, n_l, s) to $(n_i^*, n_h^*, n_l^*, s^*)$, TR, can be expressed as the product of three terms, i.e., $TR = tr_i \cdot tr_h \cdot tr_l$, where tr_i, tr_h, and tr_l are the contributions by the events from the three priority groups, respectively. The equations for the terms are given in (8), at the bottom of the next page. tr_i indicates the transition rate from n_i to n_i^* ignoring other processing elements. The equations for tr_h and tr_l are complicated depending on to which case the next state might belong where:

- number of processing elements reaches N_h or N_l;
- priority group is not granted a bus and new requests occur;
- priority group is granted a bus and new requests occur;
- service for the request is completed.

Comparisons between the base queueing model, our compromised model, and the general model are made in Fig. 5. We use single bus systems and randomly generated memory traces in the comparisons. The general model is configured with the various numbers of maximum simultaneous events 2–4. The estimation errors are computed against the trace-driven simulation results. More detailed explanation on the experimental environment is referred to Section IX-A.

Fig. 5(a) shows that both our compromised model and the general model achieve a significant improvement in accuracy compared with the base model. However, even though the general model tends to be more accurate with more simultaneous events, the accuracy improvement is not remarkable. Furthermore, as shown in Fig. 5(b), it is observed that no clear distinction exists for the variations of the estimation errors between the compromised and the general models. Fig. 5(c) represents the execution time of the linear program solver. Execution times for solving the models are about the same because the problem size by the queueing model is not dependent on the number of transitions but dependent on the number of states. However, the modeling complexity grows exponentially as the number of simultaneous events grows. Considering those observations, we confirm that our compromised model can be used effectively.

V. MODELING OF A ROUND-ROBIN-BASED BUS

The queueing model of a fixed priority bus serves as the base model for other arbitration schemes. In this section, we present the queueing model for a bus with a round-robin arbitration scheme. The round-robin arbitration is commonly used in on/off-chip bus standards with a hybrid arbitration scheme, incorporating with fixed priority or TDMA arbitration [18], [19].

In the round-robin scheme, contrary to the fixed priority scheme, the priority of each processing element dynamically varies between the highest and the lowest depending on the position of the processing element and the distance from the most recent processing element accessing a bus in the circular bus grant order. Processing elements are granted a bus in the order that their indexes are wrapped around. To construct a state transition diagram for the round-robin system, we decompose the round-robin system into multiple fixed priority systems as shown in Fig. 6 to construct a hierarchical state transition diagram. We define $BS_{\text{priority}}(N, i)$ as the fixed priority based bus system with N processing elements $(PE_0, \ldots, PE_{N-1})$ and the priorities given in the sequence of $(PE_i, \ldots, PE_{N-1}, PE_0, \ldots, PE_{i-1})$. PE_i has the highest priority. Next, we define $BS_{\text{round-robin}}(N)$ as the round-robin based bus system with N processing elements $(PE_0, \ldots, PE_{N-1})$. Therefore, we have

$$\begin{aligned} BS_{\text{round-robin}}(N) = \big\{ & bs_{\text{round-robin}}(i) \mid bs_{\text{round-robin}}(i) \\ & = BS_{\text{priority}}(N, i), \quad i = 0, \ldots, N-1 \big\}. \end{aligned}$$

A hierarchical state $bs_{\text{round-robin}}(j)$ is entered when PE_{j-1} completes a bus access. Therefore, the state transition rate from $bs_{\text{round-robin}}(i)$ to $bs_{\text{round-robin}}(j)$ is the product of the service rate of and the actual bus access rate of PE_{j-1} in the fixed priority system inside $bs_{\text{round-robin}}(i)$. If we denote $\theta_{i,j-1}$ as the actual bus access rate of PE_{j-1} inside $bs_{\text{round-robin}}(i)$, it can be computed by solving the fixed-priority-based system using the queueing model proposed in the previous section. In fact, $\theta_{i,j-1}$ is equal to θ_{j-1} of the previous section if the priority list is identically set. Since the service rate of PE_{j-1} is consistent throughout the execution, the state transition rate $\delta_{i,j}$ from $bs_{\text{round-robin}}(i)$ to $bs_{\text{round-robin}}(j)$ becomes

$$\delta_{i,j} = \theta_{i,j-1} \cdot \mu_{j-1}. \tag{9}$$

Once all transition rates are determined, we obtain the steady-state probability of each hierarchical state using an additional requirement

$$\sum_{i=0}^{N-1} P(bs_{\text{priority}}(i)) = 1.$$

Now we are ready to compute the expected wait time $w_{rr,i}$ for a bus access of PE_i on the round-robin-based system. It is the weighted sum of the expected wait time $w_{k,i}$ of PE_i in each hierarchical state $bs_{\text{round-robin}}(k)$. Thus, we arrive at the following equation:

$$w_{rr,i} = \frac{\sum\limits_{k=0}^{N-1} \left(P(bs_{\text{priority}}(k)) \cdot \theta_{k,i} \cdot w_{k,i}\right)}{\sum\limits_{k=0}^{N-1} \left(P(bs_{\text{priority}}(k)) \cdot \theta_{k,i}\right)}. \tag{10}$$

VI. Modeling of a Two-Level TDMA-Based Bus

We also use a hierarchical state diagram to model more complicated arbitration policies: modeling of a two-level TDMA-based bus is explained in this section. It consists of two arbitration schemes, primary TDMA and secondary round-robin. The

$$tr_i = \begin{cases} \mu_i, & \text{if } n_i = 1 \text{ and } n_h^* = 0 \\ 1 - \lambda_i, & \text{if } n_i = 0 \text{ and } n_h^* = 0 \\ 1 - \mu_i, & \text{if } n_i = 1 \text{ and } n_h^* = 1 \text{ and } s = \text{'}i\text{'} \\ \mu_i, & \text{if } n_i = 0 \text{ and } n_h^* = 1 \\ 1 & \text{otherwise.} \end{cases}$$

$$tr_h = \begin{cases} (1-\beta_h) \cdot \binom{N_h - n_h}{n_h^* - n_h} \cdot \gamma_h^{n_h^* - n_h} \cdot (1-\gamma_h)^{N_h - n_h^*}, & \text{if } n_h^* = N_h \text{ and } s = \text{'}h\text{'} \\ \binom{N_h - n_h}{n_h^* - n_h} \cdot \gamma_h^{n_h^* - n_h} \cdot (1-\gamma_h)^{N_h - n_h^*}, & \text{if } n_h \le n_h^* < N_h \text{ and } s \ne \text{'}h\text{'} \\ (1-\beta_h) \cdot \binom{N_h - n_h}{n_h^* - n_h} \cdot \gamma_h^{n_h^* - n_h} \cdot (1-\gamma_h)^{N_h - n_h^*} & \\ \qquad + \beta_h \cdot \binom{N_h - n_h}{n_h^* - n_h + 1} \cdot \gamma_h^{n_h^* - n_h + 1} \cdot (1-\gamma_h)^{N_h - n_h^* - 1}, & \text{if } n_h \le n_h^* < N_h \text{ and } s = \text{'}h\text{'} \\ \beta_h \cdot \gamma_h^{N_h - n_h}, & \text{if } n_h^* = n_h - 1 \\ 1, & \text{otherwise.} \end{cases}$$

$$tr_l = \begin{cases} (1-\beta_l) \cdot \binom{N_l - n_l}{n_l^* - n_l} \cdot \gamma_h^{n_l^* - n_l} \cdot (1-\gamma_l)^{N_l - n_l^*}, & \text{if } n_l^* = N_l \text{ and } s = \text{'}l\text{'} \\ \binom{N_l - n_l}{n_l^* - n_l} \cdot \gamma_l^{n_l^* - n_l} \cdot (1-\gamma_l)^{N_l - n_l^*}, & \text{if } n_l \le n_l^* \le N_l \text{ and } s \ne \text{'}l\text{'} \\ \beta_l \cdot \binom{N_l - n_l}{n_l^* - n_l + 1} \cdot \gamma_l^{n_l^* - n_l + 1} \cdot (1-\gamma_l)^{N_l - n_l^* - 1}, & \text{if } n_l \le n_l^* < N_l \text{ and } s \ne \text{'}l\text{'},\text{and } (s^* = \text{'}i\text{' or } s^* = \text{'}h\text{'}) \\ (1-\beta_l) \cdot \binom{N_l - n_l}{n_l^* - n_l} \cdot \gamma_l^{n_l^* - n_l} \cdot (1-\gamma_l)^{N_l - n_l^*}, & \text{if } n_l \le n_l^* < N_l \text{ and } s \ne \text{'}l\text{' and } s^* = \text{'}l\text{'} \\ & \qquad \text{and } (n_l^* {>} 0, \text{ or } n_l^* > 0) \\ (1-\beta_l) \binom{N_l - n_l}{n_l^* - n_l} \cdot \gamma_l^{n_l^* - n_l} \cdot (1-\gamma_l)^{N_l - n_l^* - 1} & \\ \qquad + \beta_l \cdot \binom{N_l - n_l}{n_l^* - n_l + 1} \cdot \gamma_1^{n_l^* - n_1 + 1} \cdot (1-\gamma_l)^{N_l - n_l^* - 1}, & \text{if } n_l \le n_l^* < N_l \text{ and } s = \text{'}l\text{' and } s^* = \text{'}l\text{'} \\ & \qquad \text{and } n_h^* = 0, \text{ and } n_l^* = 0 \\ \beta_l \cdot \gamma_l^{N_l - n_l}, & \text{if } n - h^* = n_h - 1, \\ 1 & \text{otherwise} \end{cases} \tag{8}$$

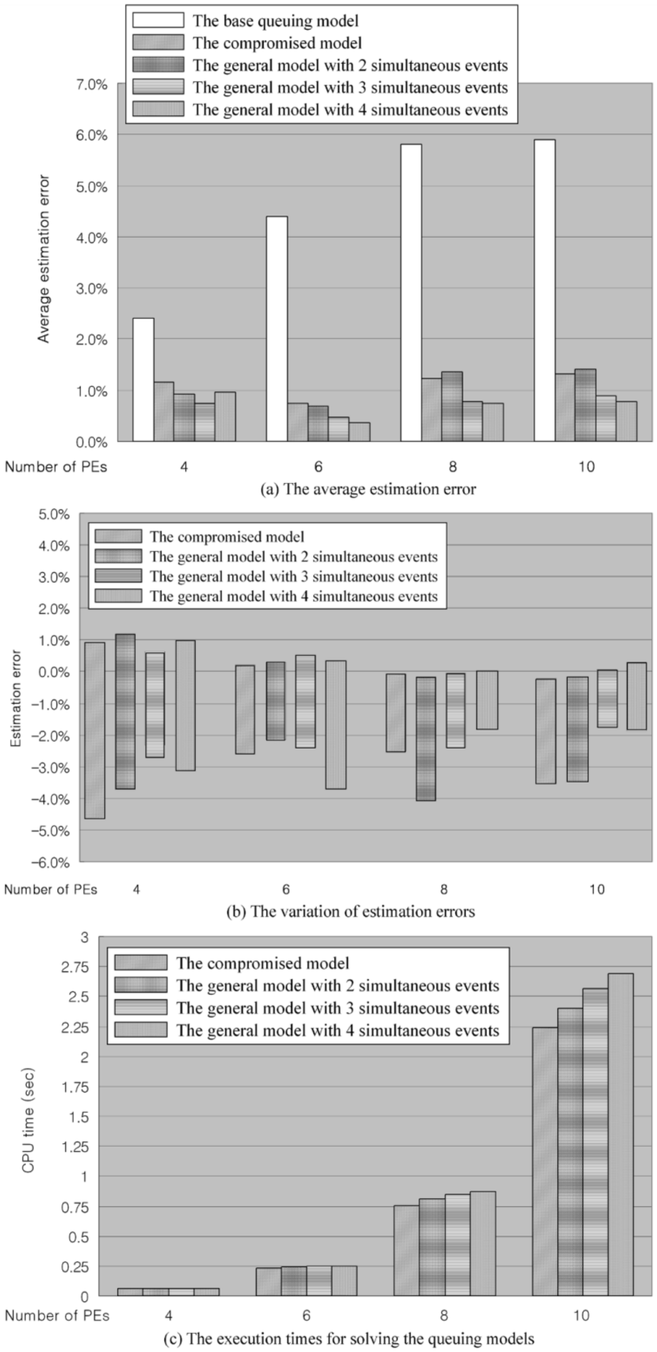


Fig. 5. Comparisons between the compromised queueing model and the general queueing models with the various numbers of simultaneous events.

first level of arbitration uses a timing wheel where each slot is statically reserved for a unique master. A processing element with heavy communication requirements may reserve more than one slot. Although the TDMA-based architecture guarantees a fixed bus bandwidth for each processing element, no bus request from the processing element associated with the current slot means the waste of bus bandwidth. In order to prevent the waste of this unused slot, another processing element may be granted the bus by round-robin arbitration during the slot.

A two-level TDMA system can also be treated as the composition of multiple fixed priority systems similarly to the round-robin based system discussed in the previous section.

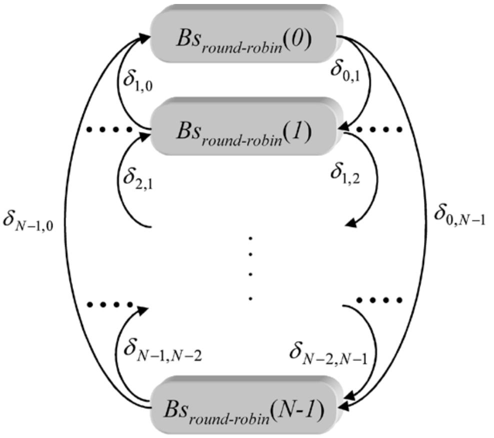


Fig. 6. State transition diagram of a round-robin arbitration bus.

Consider the example system that has four processing elements $(PE_0, \ldots, PE_3)$. When the total bandwidth of a bus is normalized to 1, PE_i is assigned a bandwidth $bw(i)$ less than 1 and the sum of the bandwidths of all processing elements becomes 1. Suppose that PE_0 is assigned the current TDMA slot and PE_1 has the highest priority in the round-robin order. The bus grant order for round-robin arbitration is again assumed to be the order of processing element indexes. Therefore, the priority at the current time slot will be (0,1,2,3), i.e., PE_0 has the highest priority, PE_1 becomes the second, and so on.

More generally, we define an assignment sequence of time slots as follows:

$$TS(N) = \{ts(i) \mid i = 0, \ldots, N-1, \sum_{i=0}^{N-1} bw(i) = 1\}$$

where N processing elements are assigned and $ts(i)$ is a set of time slots that PE_i is assigned. We also define the two-level TDMA based bus system as

$$\begin{aligned} &BS_{\text{2level-TDMA}}(N) \\ &\quad = \{bs_{\text{2level-TDMA}}(i,j) \mid bs_{\text{2level-TDMA}}(i,j) = ts(i) \\ &\qquad \cup\, bs_{\text{priority}}(N1,j),\ i = 0, \ldots, N-1; \\ &\qquad j = 0, \ldots, N-1\} \end{aligned}$$

where $bs_{\text{2level-TDMA}}(i,j)$ is the hierarchical state in which PE_i is assigned the current time slot and PE_j has the highest priority in the priority based system with $N-1$ processing elements except for PE_i. Therefore, the overall priority order in $bs_{\text{2level-TDMA}}(i,j)$ becomes $(PE_i, PE_j, \ldots, PE_{N-1}, PE_0, \ldots, PE_{i-1})$.

Fig. 7(b) shows the example state transition from $bs_{\text{2level-TDMA}}(0,1)$ for a given fixed 16-slot assignment of the TDMA protocol as shown in Fig. 7(a). If PE_0 is granted a bus in the state $bs_{\text{2level-TDMA}}(0,1)$, the next state is one of $bs_{\text{2level-TDMA}}(0,1)$, $bs_{\text{2level-TDMA}}(1,2)$, $bs_{\text{2level-TDMA}}(2,1)$, and $bs_{\text{2level-TDMA}}(3,1)$ depending upon which processing element gets the next TDMA slot. Overall 12 transitions can be drawn from $bs_{\text{2level-TDMA}}(0,1)$ as shown in Fig. 7(b). But some destination states are duplicated in the figure. For instance, the transition from $bs_{\text{2level-TDMA}}(0,1)$ to $bs_{\text{2level-TDMA}}(1,2)$ can occur by granting the bus to either PE_0 or PE_1. On the other hand, some state transitions are not allowed in the given slot assignment. For example, when PE_0 is assigned the cur-

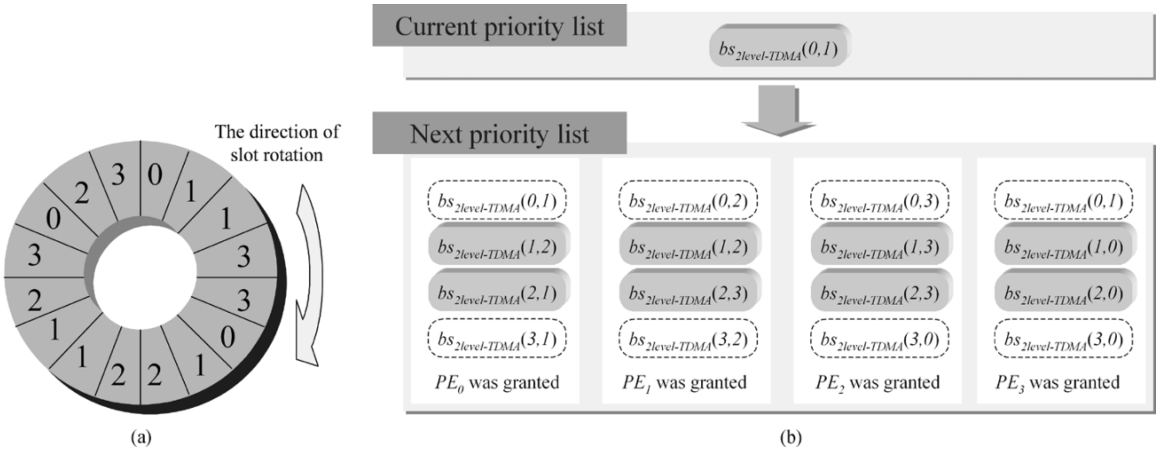

The direction of slot rotation
2
3
0
1
1
3
3
0
1
2
2
1
1
2
3
0
Current priority list
$bs_{2level\text{-}TDMA}(0,1)$
Next priority list
$bs_{2level\text{-}TDMA}(0,1)$
$bs_{2level\text{-}TDMA}(1,2)$
$bs_{2level\text{-}TDMA}(2,1)$
$bs_{2level\text{-}TDMA}(3,1)$
PE_0 was granted
$bs_{2level\text{-}TDMA}(0,2)$
$bs_{2level\text{-}TDMA}(1,2)$
$bs_{2level\text{-}TDMA}(2,3)$
$bs_{2level\text{-}TDMA}(3,2)$
PE_1 was granted
$bs_{2level\text{-}TDMA}(0,3)$
$bs_{2level\text{-}TDMA}(1,3)$
$bs_{2level\text{-}TDMA}(2,3)$
$bs_{2level\text{-}TDMA}(3,0)$
PE_2 was granted
$bs_{2level\text{-}TDMA}(0,1)$
$bs_{2level\text{-}TDMA}(1,0)$
$bs_{2level\text{-}TDMA}(2,0)$
$bs_{2level\text{-}TDMA}(3,0)$
PE_3 was granted
(a)
(b)

Fig. 7. (a) An example of TDMA slot assignment and (b) the state transitions from $bs_{2\text{level-}\mathrm{TDMA}(0,1)}$ according to the slot assignment in (a).

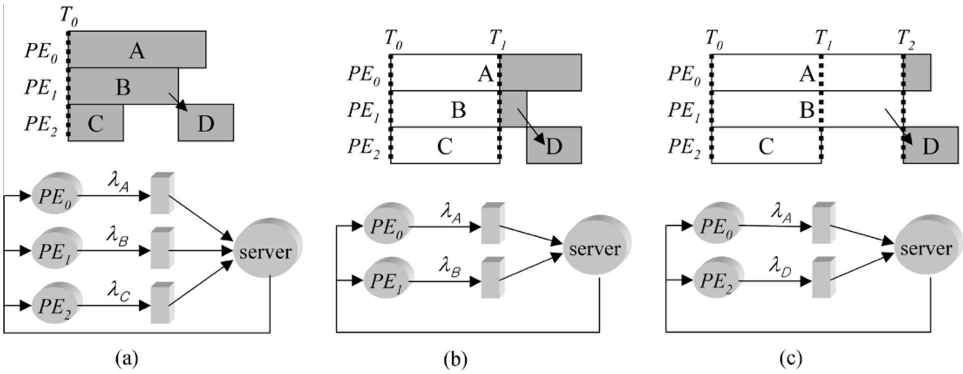


Fig. 8. (a) An example schedule of PE_0 and PE_1, and PE_1 and its corresponding queueing model, (b) new queueing model after the function block C is finished at T_1, and (c) another queueing model after the function block B is finished at T_2.

rent slot in Fig. 7(b), the next slot can be allocated only to PE_1 and PE_2 so that no transitions to $bs_{2\text{level-}\mathrm{TDMA}}(0,x)$ and to $bs_{2\text{level-}\mathrm{TDMA}}(3,x)$ can be made. Those infeasible transitions are represented as the dashed lines in Fig. 7(b). In general, for given N processing elements, there are less than $N\cdot(N-1)$ state transitions from a certain state.

We define $b_{i,m}$ as the probability that PE_m can be assigned the next TDMA slot when PE_i is assigned the current slot. In the case of Fig. 7(a), $b_{0,1}$ and $b_{0,2}$ become 0.67 and 0.33, respectively. Both $b_{0,0}$ and $b_{0,3}$ are zero. Suppose that the destination state is $bs_{2\text{level-}\mathrm{TDMA}}(m,n)$ when PE_k is granted a bus in $bs_{2\text{level-}\mathrm{TDMA}}(i,j)$. Then the transition rate from $bs_{2\text{level-}\mathrm{TDMA}}(i,j)$ to $bs_{2\text{level-}\mathrm{TDMA}}(m,n)$, $\delta_{(i,j)\to(m\cdot n)}$, becomes the product of the actual bus request rate of PE_k in $bs_{2\text{level-}\mathrm{TDMA}}(i,j)$, the service rate of PE_k, and $b_{i,m}$ so that we get the following equation:

$$\delta_{(i,j)\to(m,n)} = b_{i,m}\cdot\theta_{k,(i,j)}\cdot\mu_k \tag{11}$$

where $\theta_{k,(i,j)}$ is the actual bus request rate of PE_k in $bs_{2\text{level-}\mathrm{TDMA}}(i,j)$ and can be obtained by solving the fixed priority system where PE_i and PE_j have the highest and the second priorities respectively as explained in Section IV. We obtain the steady state probability of each $bs_{2\text{level-}\mathrm{TDMA}}(i,j)$ with these state transitions rates and an additional requirement:

$$\sum P(bs_{2\text{level-}\mathrm{TDMA}}(i,j)) = 1.$$

A similar equation as (10) can be used to compute the expected wait time for each processing element.

VII. Static Estimation Using Schedule Information

A. Estimation of Single Bus Systems

This section explains how the task schedule information is used in our estimation method. A simple statistical modeling of the bus requests from a processing element assumes that the bus requests are distributed evenly throughout the whole execution duration of an application. This assumption is one of the main sources of the inaccuracy of simple static statistical modeling. Since we assume that the schedule of function blocks is predetermined, the pattern of bus requests is determined statically as shown in Fig. 8. Note that the initial schedule is made without considering the wait time for bus access. We divide the schedule into several time slots in such a way that all processing elements maintain their bus request patterns during each time slot. Then, we apply the proposed queueing analysis in each time slot starting from the beginning of schedule. The shaded regions in the figure indicate the remaining sections for the static estimation.

During the first time slot, three function blocks A, B, and C are executed concurrently on PE_0, PE_1, and PE_2 respectively. To evaluate the expected wait delay for bus access from each processing element, a queueing system is constructed as shown in Fig. 8(a). From the proposed queueing analysis, we compute how much the initial schedule length of function blocks is extended due to bus contention. Suppose that the function block C on PE_2 is finished first at T_1. Then we consider the next time

```
Estimate_Single_Bus(B, MT, ORI_SCHED, PL, FL, EVAL_SCHED)
begin
  EVAL_SCHED = ORI_SCHED
  cur_time = 0
  for each $pe \in PL$ do
    pe.cur_fb=NULL
    pe.cur_fb_done=FALSE
  end for
  while (FL is not empty) do
    for each $pe \in PL$ do
      if (pe.cur_fb_done==TRUE or pe.cur_fb==NULL) then
        pe.cur_fb = Get_Current_Fb(EVAL_SCHED, PL, FL)
        stat_params = Get_Stat_Params(ORI_SCHED, PL, FL, MT)
        if (pe.fb_start_time < cur_time) then
          cur_time = pe.fb_start_time
        end if
      end if
    end for
    cur_time = Estimate_End_Time(B,PL,EVAL_SCHED,stat_params)
    Update_Schedule(EVAL_SCHED, cur_time)
  end while
end Estimate_Single_Bus
```

Fig. 9. Schedule-aware performance estimation algorithm for a single bus.

slot where two function blocks A and B are accessing a bus as shown in Fig. 8(b) and construct another queueing model with PE_0 and PE_1. This time slot is also lengthened until the function B is completed at T_2. After T_2, another queueing model with PE_0 and PE_2 is made for the evaluation of the remaining schedule. This evaluation process is repeated until the queueing model examines all of the scheduled function blocks.

The overall algorithm of the estimation technique for a single bus is described in Fig. 9. The procedure **Estimate_Single_Bus** has five inputs. B represents the data structure of a single bus. *MT* is the memory traces of all processing elements and *ORI_SCHED* is the initial schedule. *PL* is a set of processing elements and *FL* is a set of function blocks used. The output, *EVAL_SCHED*, of this procedure is the updated schedule after considering all bus conflicts and any other overheads.

The main procedure of the queueing analysis is **Estimate_End_Time**. Before the queueing analysis, we first determine which processing elements request a bus by calling the procedure **Get_Current_Fb** and compute the statistical parameters by calling the procedure **Get_Stat_Params**. We compute two statistical parameters of each function block from the memory traces *MT*: they are the memory access rate λ with no bus conflicts and the mean service time S. The memory access rate during the execution of the function block *FB* on the processing element *PE* is formulated as follows:

$$\lambda_{FB,PE} = \frac{M_{FB,PE}}{EXE_{FB,PE}} \tag{12}$$

where $M_{FB,PE}$ is the total memory access counts and $EXE_{FB,PE}$ is the execution time of the function block FB on the processing element PE. When computing the mean service time, we consider the different burst transfer size according to the memory access type. For example, code memory access is usually the burst access of which the size equals to the cache line size, whereas data memory access may have various burst lengths. Therefore, we define a set of memory access types AT with three types: code memory access $AccCode$, data memory access $AccData$, and shared memory access $AccShared$

$$AT = \{AccCode, AccData, AccShared\}.$$

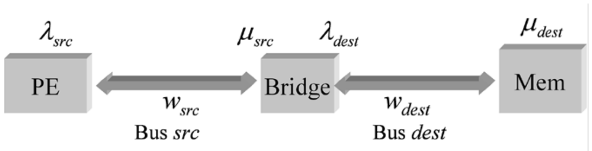


Fig. 10. The modeling of communication via a bus bridge.

For each set of access types, we further define a set of burst transfer types BT according to the burst transfer length:

$$BT = \{single, four\text{-}beat, eight\text{-}beat, random\}.$$

There may be more burst transfer types in general. However, for simplicity, we assume only 4 types of burst transfers, which are used in the ARM720T processor. Then, the mean service time for the function block FB on the processing element PE is computed as

$$S_{FB,PE} = \sum_{at \in AT} \left(\sum_{bt \in BT} S_{at,bt,PE} \cdot \frac{M_{at,bt,FB,PE}}{M_{FB,PE}} \right), \tag{13}$$

where $S_{at,bt,PE}$ and $M_{at,bt,FB,PE}$ are the service time including the bus overhead as well as the memory access time and the burst transfer counts of the type bt in the memory access type at for the function block FB allocated on the processing element PE respectively.

The final procedure **Update_Schedule** modifies the initial schedule to obtain the updated schedule, *EVAL_SCHED*, after the current time slot. From the updated schedule, we define the next time slot and go back to the main iteration body until all function blocks are considered. In the case that the number of function blocks is huge or the schedule length is very long, the time complexity of our proposed technique approaches to that of the trace-driven simulation due to too frequent invocation of linear program solver. To reduce this problem, neighboring function blocks can be clustered into a group, which is discussed in Section VIII-C.

B. Extension to Multiple Bus Systems

Communications across buses are achieved via a bus bridge in multiple bus systems. A bus bridge plays both roles of a processing element and a memory as displayed in Fig. 10. We assume for simple analysis that no communication passes through more than 3 buses.

Fig. 10 shows how the bridge is modeled when a processing element on the bus *src* accesses the memory *Mem* on the bus *dest*. The request rate λ_{src} of the processing element PE is reflected to the bus $dest$ by the request rate λ_{dest} of the bridge. We have to know the expected waiting time w_{src} of the processing element PE on the bus src to compute λ_{dest}. At the same time, the bridge looks like a memory from the bus src point of view. Therefore, the service rate μ_{src} of the bridge should be computed. Let w_{dest} be the expected waiting time of the bridge on the bus $dest$. Then, λ_{dest} and μ_{src} are computed as follows:

$$\begin{aligned} \frac{1}{\lambda_{dest}} &= \frac{1}{\lambda_{src}} + w_{src} + O_{bridge} \\ \frac{1}{\mu_{src}} &= w_{dest} + O_{bridge} + \frac{1}{\mu_{dest}} \end{aligned} \tag{14}$$

where O_{bridge} is the overhead associated with the bridge. On the other hand, w_{src} and w_{dest} are obtained from our estimation

TABLE II
COMPLEXITY OF THE PROPOSED ESTIMATION TECHNIQUE

The number of	Fixed priority			Round-robin	Two-level TDMA
processing elements	CPU time	Min. Num. of states	Max. Num. of states	CPU time	CPU time
2	0.04 sec	5	5	0.04 sec	0.04 sec
4	0.11 sec	21	21	0.21 sec	0.44 sec
6	0.34 sec	37	47	1.22 sec	5.16 sec
8	0.92 sec	53	83	5.62 sec	32.16 sec
10	2.68 sec	69	129	21.05 sec	167.59 sec

technique after computing λ_{dest} and μ_{src}. Therefore, the estimation and the bridge modeling are performed iteratively until all parameters become stable.

VIII. Time Complexity

As explained in the previous section, our proposed technique progresses dividing the entire schedule into several time slots in which separate queueing models are constructed and to which the proposed queueing analysis is applied. Therefore, the time complexity of the proposed technique depends on the product of the time complexity of the queueing analysis and the number of time slots. First, we consider the time complexity of the queueing analysis.

Table II shows the run time of the proposed estimation technique and the maximum and the minimum number of states varying the number of processing elements on Xeon 2.8 GHz workstation running Linux. We used a GNU Scientific Library (GSL) [17] to solve the linear equations of the proposed method. In the case of the fixed priority base system, for each processing element PE_i, the number of states $\{(n_i, n_h, n_l, s)\}$ is $O(N^2)$ where N is the total number of processing elements. The number of states depends on the priority of the processing element of interest. Since the time complexity of linear program solver is pseudo-polynomial, the overall time complexity is also pseudo-polynomial.

As for the round-robin based system, since there exist N priority lists for N processing elements, the complexity becomes roughly N times larger than that of the fixed priority based system. On the other hand, since the two-level TDMA based system with N processing elements may have up to $N \cdot (N-1)$ priority lists, its complexity is roughly $N \cdot (N-1)$ times larger than the fixed priority case. Note that all three cases still have the pseudo-polynomial time complexity although the overall complexity increase as the arbitration scheme becomes complicated. Table II confirms that the proposed technique has the acceptable complexity for fast design space exploration.

A. Comparison With Trace-Driven Simulation

While the time complexity of the proposed technique depends on the number of processing elements as discussed above, that of the trace-driven simulation depends on the trace size as well as the number of processing elements. Considering those parameters, comparisons are made quantitatively by measuring the execution time as shown in Fig. 11. The number of processing elements is varied from 2 to 16. The average schedule length of each processing element is also varied from 10^5 cycles to 10^7 cycles. The average bus request rate for each processing element is set to about 0.12, which is a typical value for multimedia applications.

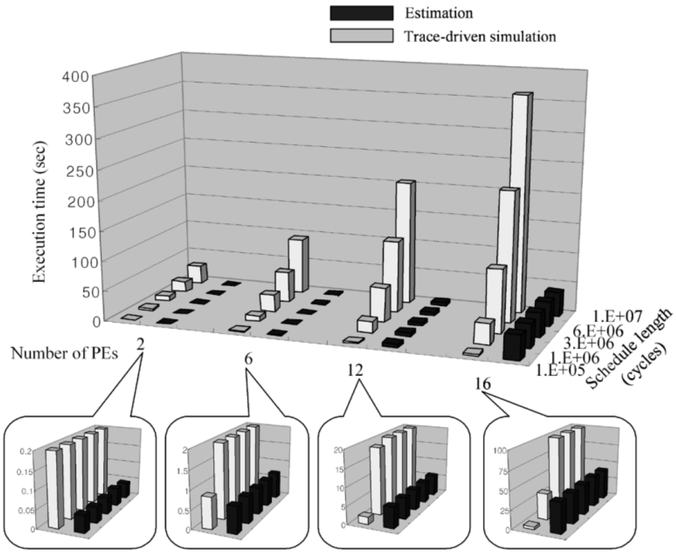


Fig. 11. Comparison of time complexity between the proposed technique and a trace-driven simulation technique.

The time complexity of the proposed technique increases faster than the trace-driven simulation as the number of processing elements increases. It explains why the time complexity of the proposed technique is larger when the schedule length is short and the number of processing elements is 16 in Fig. 11. However the execution time of the trace-driven simulation grows proportionally to the trace size while that of the proposed approach remains the same. Therefore the benefit of using the proposed technique grows as the length of the time slot increases.

B. Effects of Schedule Complexity

The time complexity of the proposed technique is also proportional to the number of time slots. Since each time slot is defined when a function block finishes its execution, the number of time slots is about the same as the number of scheduled function block invocations. In many video applications, the number of function block invocations can be very large due to the different execution rates of function blocks. Fig. 12 shows an H.263 decoder specification example that is mapped to two processing elements: an ARM9 processor and a dedicated hardware for IDCT as specified in Fig. 12(a) and (b), respectively. To decode one macro block, one invocation of gray function blocks and four invocations of black function blocks are required. To decode one QCIF-formatted frame that consists of 99 macro blocks, the schedule contains the total 6968 invocations of the function blocks.

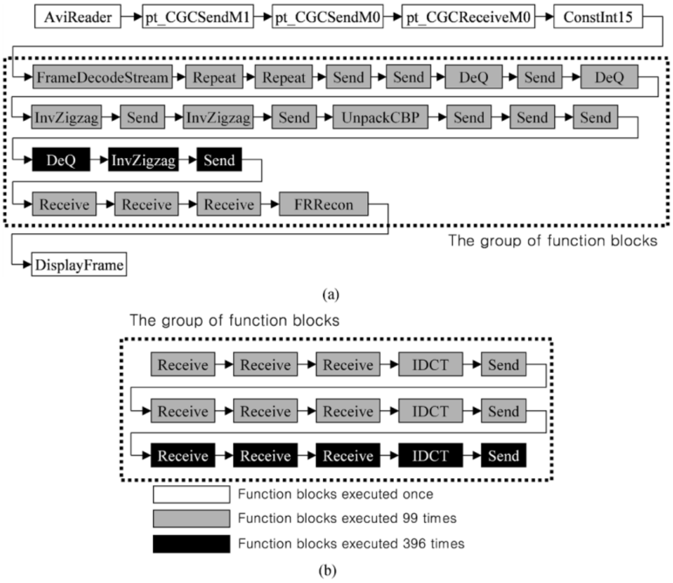


Fig. 12. The specification of H.263 Decoder and the mapping of function blocks to: (a) an ARM9 processor and (b) a dedicated IDCT hardware. The groups in (a) and (b) are executed 99 times during decoding one frame.

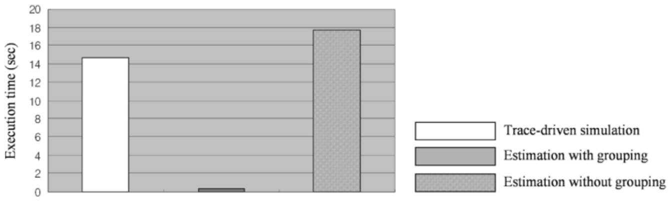


Fig. 13. Grouping the function blocks in the schedule is required to enhance the efficiency of the proposed technique.

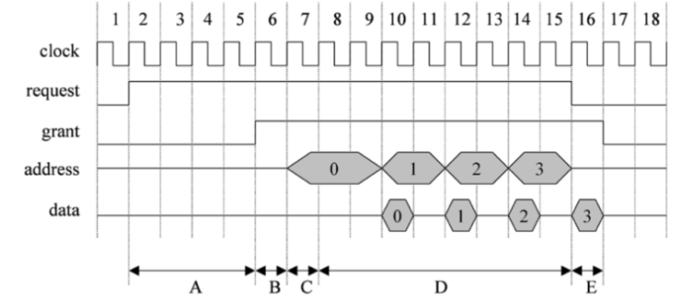
C. Effects of Schedule Complexity

Fig. 13 shows that the naïve application of the proposed technique is worse than the trace-driven simulation since the average granularity of the function blocks is too small. Therefore, we group the function blocks that are repeatedly executed in the same sequence. Such grouping also reduces the effective number of time slots. In this example, we group the function blocks that decode one macro block and apply the proposed estimation technique to the first invocation of the group. A significant efficiency gain, about 47 times, is obtained by grouping the function blocks as shown in the figure.

IX. Experiments

A. Estimation Accuracy of Single Bus Systems

To investigate the accuracy of the proposed static estimation method over a wide variety of working conditions, the first set of experiments considers a single bus architecture varying number of processing elements, bus request rates, bus service rates (or memory access times), and burst lengths. Then we compared the estimation results with those obtained from the trace-driven simulation. Our trace-driven simulator adjusts the time stamps of trace data by accurately modeling the communication architecture that includes buses and memories. More precisely, in these experiments, the bus model used in our trace-driven simulator consists of 4 phases: *bus-arbitration*, *start-address-drive*, *sequential-burst-transfer*, and *last-data-drive*.



A: Waiting time for the use of a bus (dependent on bus conflicts)

B: Bus arbitration

C: Start address drive

D: Sequential burst transfer (consists of burst initialization and sequential accesses)

E: Last data drive

Fig. 14. Burst transfer of four words on our bus model.

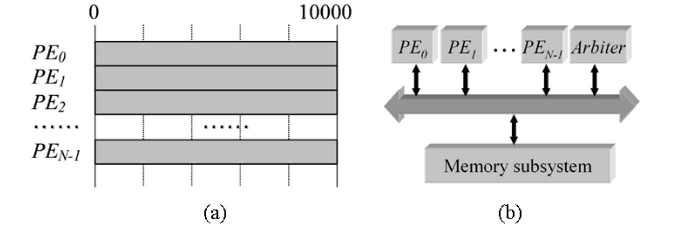
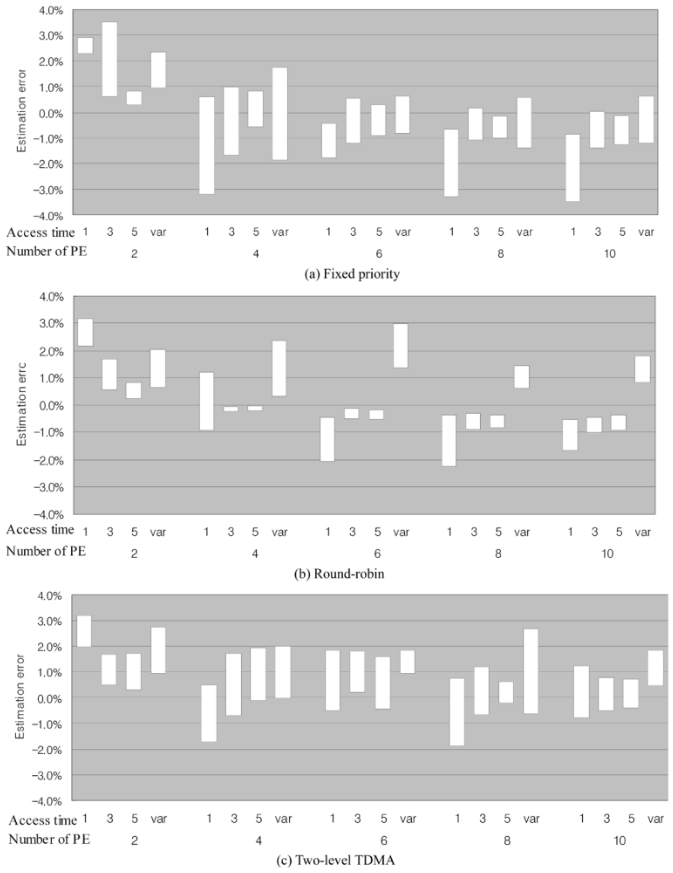


Fig. 15. (a) Schedule of function blocks on N processing elements $(PE_0, PE_1, \ldots, PE_{N-1})$ and (b) a single bus architecture that consists of N processing elements and a memory subsystem.

Fig. 14 shows the burst transfer of four words complying with this bus model. The part tagged with 'A' means the waiting time of a processing element due to bus conflicts. The parts B, C, and E are the bus protocol specific overheads and are all set to 1 cycle in this experiment. The part D consists of more than one word accesses to the memory. The memory is configured with two parameters, which are the initialization cycle for burst transfer and the access cycle taken for a single word during the burst transfer. For example, they are set to 1 and 2 clock cycles respectively in Fig. 14. It is assumed that other control signals are synchronized with the address bus and are not shown here for simplicity.

A template example system for our first set of experiments is described in Fig. 15. N processing elements are selected and all processing elements are busy during the same schedule length, 10 000 cycles. In order to maximize bus conflicts, all processing elements are connected to a single bus as shown in Fig. 15(b). The number of processing elements and the memory subsystem access time define the configuration points in this architecture template. We vary the number of processing elements from 2 to 10. The bus request rate λ of each processing element is chosen randomly within the range from 0.05 to 0.2. For a selected bus request rate, we generated the memory traces from each processing element following the Poisson distribution (i.e., the exponentially distributed inter-arrival times between bus requests). While generating the memory traces, we also randomized the burst lengths. As for the memory subsystem, the burst initialization overhead is always fixed to 1 cycle. The single word access time of each processing element is assigned in four different ways: the identical assignments of 1, 3, and 5 cycles to



(a) Fixed priority

(b) Round-robin

(c) Two-level TDMA

Fig. 16. Estimation accuracy according to the number of processing elements and the memory access time for each arbitration scheme.

all processing elements and the random assignments, among 1, 3, and 5 cycles, to each processing element. In the case that λ is 0.1 and memory access time is 1 cycle, the portion of the total memory access time over the entire execution time is about 0.36. This portion is almost same as the case of an H.263 encoding algorithm, which will be discussed later.

Using these configuration points, 20 example systems were constructed for each of three arbitration schemes: fixed priority, round-robin, and two-level TDMA. We performed the proposed static analysis and the trace-driven simulation to compare the total execution time of the example systems. For each set of experiments, we generated the memory traces 10 times. Fig. 16 shows the experimental results, which indicate the range of estimation errors of the completion times. It is observed that the estimation error of the proposed method does not exceed 4% in all cases. These experiments show the robustness of the proposed technique on various architecture configurations and arbitration schemes.

B. Comparison With Simpler Models

In this section, we show the improvement of our proposed model over two simpler estimation models: the base queueing model of an I/O bus reviewed in Section IV-A and the intuitive analytical model assuming a fixed priority bus system. It is questionable whether there is a simpler static estimation method that is reasonably accurate. So, we devised an intuitive equation on the expected waiting time of PE_i, w_i, by bus contention, as shown in (15) for a single bus that has N processing elements

$$w_i = \frac{\sum_{j=0}^{i-1} m_j}{exe_i} \cdot \frac{S_i}{2} \cdot (i-1) + \frac{\sum_{j=i+1}^{N-1} m_j}{exe_i} \cdot \frac{S_i}{2} \tag{15}$$

where m_j, exe_j, and S_j are the total memory access time, the total execution time, and the mean service time of PE_j respectively. At the right side of (15), the first term is the expected waiting time due to the bus requests from the priority group Φ_h and the second term is the waiting time due to the current outstanding request of the priority group Φ_l.

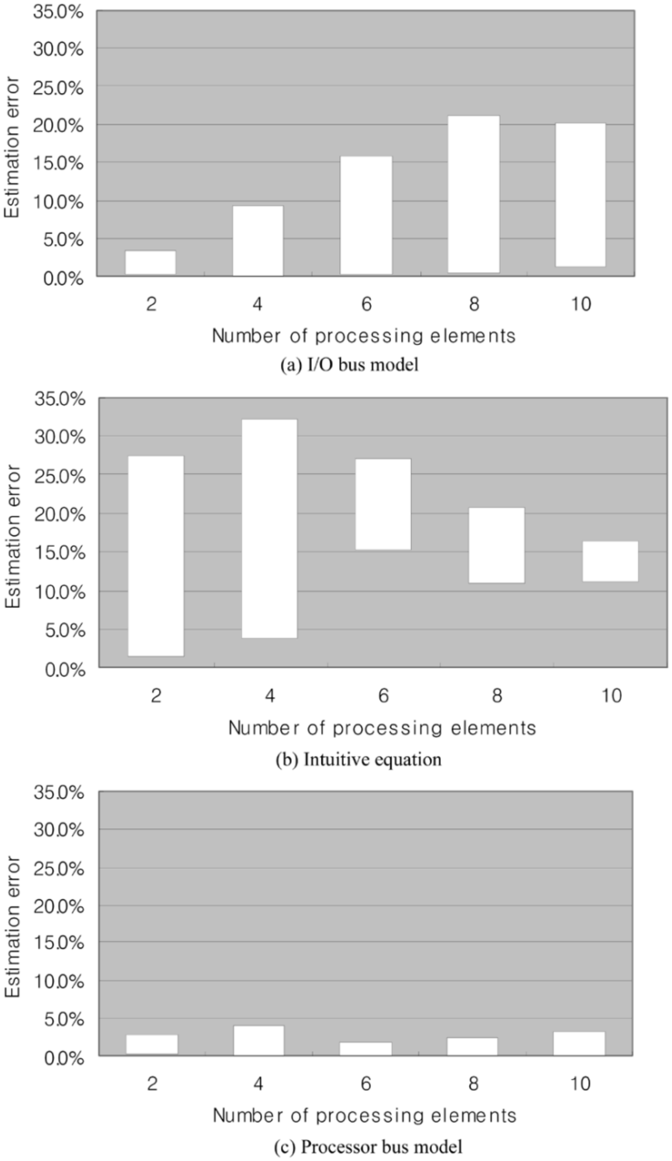
The same experimental environment in Fig. 15 is used again. We estimated the performance of 20 communication architecture configurations using the base model of an I/O bus and (15). Each experiment was repeated 10 times with randomly generated memory traces. Fig. 17(a)–(c) summarize the estimation results by three estimation techniques in terms of the absolute error range compared with the trace-driven simulation. One bar graph for a given number of processing elements covers all kind of the memory access times mentioned in the previous section.

Although the average estimation error of the base queueing model does not exceed 6% as shown in Fig. 17(d), the variation of errors becomes larger rapidly as the number of processing elements increases. The intuitive equation does not produce the acceptable estimation accuracy on the variation of errors as well as the average error rate. On the other hand, the processor bus model shows the consistent estimation errors within the range of about 1.3% on average.

C. Design Space Exploration of 4-Channel DVR

Next, we validate our proposed technique by applying it to a practical example, 4-channel digital video recorder (*DVR*). The raw bit streams from external four sources are encoded separately by *DVR* using an H.263 encoding algorithm. Fig. 18(a) shows the specification of an H.263 encoding algorithm. All function blocks of an H.263 encoder except for the motion-estimation (*ME*) and the discrete-cosine-transform (*DCT*) blocks are mapped to one ARM720T. And all ME and *DCT* blocks are mapped to the dedicated hardware blocks for *ME* and *DCT*, respectively, so that four H.263 encoders share two hardware blocks. Therefore the initial schedule of each function block is constructed like Fig. 18(b). Fig. 18(c) summarizes the total memory access counts for each processing element. Memory traces are obtained by encoding a P-frame of QCIF-formatted bit-stream from the same video clip for all channels.

With the *DVR* example, we verified our proposed exploration flow explained in Section II. Many architecture candidates are generated changing the number of buses, the mapping of processing elements to buses, the move of related shared memory segments, and the associated bridge connections. Other communication architecture parameters, such as priority assignment, bus data-width, and so on, are fixed to the arbitrary values [20].



(a) I/O bus model

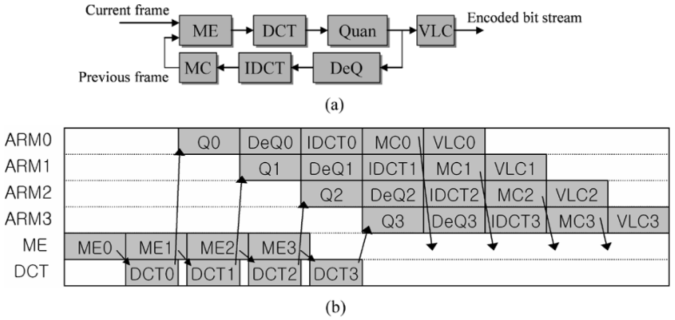
(b) Intuitive equation

(c) Processor bus model

Estimation technique	Number of processing elements				
	2	4	6	8	10
I/O bus model	1.4 %	2.4 %	4.4 %	5.8 %	5.9 %
Intuitive equation	16.6 %	26.3 %	22.6 %	18.5 %	14.0 %
Processor bus model	1.3 %	1.1 %	0.5 %	0.6 %	1.0 %

(d) Average estimation errors

Fig. 17. The comparison of estimation errors for the example systems of Fig. 15 between: (a) the base queueing model for an I/O bus, (b) the intuitive equation, and (c) the proposed model for the processor bus. (d) Average estimation errors according to the estimation techniques.



Processing element	Access counts
ARM0, ARM1, ARM2, ARM3	1,422,860
ME	139,392
DCT	304,128

(c)

Fig. 18. (a) The specification of an H.263 encoding algorithm, (b) the initial schedule of 4-channel digital video recorder *(DVR)*, and (c) the size of memory traces.

TABLE III
EXPERIMENTAL RESULTS ABOUT 4-CHANNEL *DVR* FOR THREE ARBITRATION SCHEMES

Arbitration scheme	Number of buses	Number of explored architectures	Number of included bridges		Error (%)	
			Range	Avg.	Range	Abs. Avg.
Fixed priority	1	1	0~0	0.0	-5.17~-5.17	5.17
	2	45	1~1	1.0	-5.96~5.60	2.56
	3	59	2~3	2.8	-0.28~8.45	4.68
	4	69	3~5	4.7	-0.10~7.45	2.24
	5	183	4~9	7.7	-0.20~7.45	1.15
	6	50	9~12	10.8	-0.12~0.97	0.18
Round-robin	1	1	0~0	0.0	0.18~0.18	0.18
	2	45	1~1	1.0	-0.64~5.05	2.94
	3	59	2~3	2.7	3.69~7.22	5.65
	4	69	3~5	4.7	-0.13~7.28	2.20
	5	181	4~9	7.6	-0.20~6.75	1.10
	6	50	9~12	10.8	-0.12~0.97	0.18
2-Level TDMA	1	1	0~0	0.0	-0.11~-0.11	0.11
	2	45	1~1	1.0	0.02~5.05	2.90
	3	59	2~3	2.7	4.28~6.96	5.54
	4	114	3~6	4.9	-0.04~7.09	1.96
	5	71	4~9	7.5	-0.08~6.53	0.67
	6	10	11~11	11.0	0.01~0.99	0.19

The results of exploration are summarized in Table III. The number of buses is changed from 1 to 6. For a given number of buses, the number of bus bridges reflects indirectly the complexity of bus topology. In the last two columns on the right side, the estimation errors compared with the trace-driven simulation are reported with its range and the average absolute values. During the exploration over more than 200 architectures for each arbitration scheme, our proposed estimation technique keeps its accuracy within the range from -6% to 8.5%. The time for the trace-driven simulation is about 5 min for one bus architecture with fixed priority arbitration while the proposed scheme spends about 1.5 s. Through this experiment, we verified the validity of our proposed static estimation techniques.

X. CONCLUSION

In this paper, we presented an efficient static estimation technique of bus architectures based on three commonly used arbitration schemes: fixed priority, round-robin, and two-level TDMA. It is based on the queueing model and makes use of the schedule information and the memory traces. Since the estimation time is pseudo-polynomial to the number of processing elements, the number of function blocks, and the number of buses used, the proposed technique can be used to prune the large design space before the trace-driven simulation. Experimental results show that our proposed technique is several orders of magnitude faster than the trace-driven simulation while keeping the estimation error within 8% consistently in the various communication architecture configurations.

The bus architectures we assume in this paper do not have the advanced features being used in the-state-of-art bus architectures such as split-transaction, multiple outstanding bus masters, out-of-order transaction, and so on. Such advanced features will affect the service rate of the memory system in our simplified queueing model. It will be a future research topic to investigate the scope of applicability of the proposed technique. Since the objective of the proposed static analysis is to reduce the design space, a reasonable amount of inaccuracy is still tolerable to accelerate the design space exploration.

So far, we assumed that the schedule of function blocks is known *a priori* before static estimation. In a more general case of multi-task environment, the schedule can be dynamically varying. We are currently extending the estimation technique to accommodate such dynamically varying cases conservatively.

Acknowledgment

The authors would like to acknowledge the useful comments and suggestions of the anonymous reviewers who helped improve the quality of this paper.

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 Integr. Circuits Syst.*, vol. 19, no. 12, pp. 1523–1543, Dec. 2000.

[2] A. Brandwajn, "A note on SCSI bus waits," *ACM SIGMETRICS Perf. Eval. Rev.*, vol. 30, pp. 41–47, Sep. 2002.

[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*, Oct. 1999, pp. 181–190.

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

[5] K. Hines and G. Borriello, "Optimizing communication in embedded system cosimulation," in *Proc. Int. Symp. Hardware/Software Codesign*, Mar. 1997, pp. 121–125.

[6] P. V. Knudsen and J. Madsen, "Communication estimation for hardware/software codesign," in *Proc. Int. Symp. Hardware/Software Codesign*, Mar. 1998, pp. 55–59.

[7] P. Knudsen and J. Madsen, "Integrating communication protocol selection with partitioning in hardware/software codesign," in *Proc. Int. Symp. System Level Synthesis*, Dec. 1998, pp. 111–116.

[8] T. Yen and W. Wolf, "Communication synthesis for distributed embedded systems," in *Proc. Int. Conf. Computer Aided Design*, Nov. 1995, pp. 288–294.

[9] R. B. Ortega and G. Borriello, "Communication synthesis for distributed embedded systems," in *Proc. Int. Conf. Computer Aided Design*, Nov. 1998, pp. 437–444.

[10] A. Nandi and R. Marculescu, "System level power/performance analysis for embedded systems design," in *Proc. Int. Conf. Design Automation*, Jun. 2001, pp. 599–604.

[11] J. Daveau, T. B. Ismail, and A. A. Jerraya, "Synthesis of system-level communication by an allocation based approach," in *Proc. Int. Symp. System Level Synthesis*, Sep. 1995, pp. 150–155.

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

[13] Synopsys, CoCentric System Studio.. [Online]. Available: http://www.synopsys.com/products/cocentric_studio/cocentric_studio.html

[14] CoWare, N2C Design System.. [Online]. Available: http://www.coware.com/cowareN2C.html

[15] Mentor Graphics. Seamless CVE. [Online]. Available: http://www.mentorg.com/seamless/

[16] S. Stidham, "A last word on $L = \lambda W$," *Oper. Res.*, vol. 22, pp. 417–421, 1974.

[17] GNU Scientific Library (GSL), GNU and FSF. [Online]. Available: http://www.gnu.org/software/gsl

[18] Sonics. Sonics Integration Architectures. [Online]. Available: http://www.sonicsinc.com

[19] PCI-SIG, PCI (Peripheral Component Interconnect).. [Online]. Available: http://www.pcisig.com

[20] S. Kim, C. Im, and S. Ha, "Efficient exploration of on-chip bus architectures and memory allocation," in *Proc. Int. Conf. Hardware/Software Codesign and System Synthesis*, Sep. 2004, pp. 248–253.

[21] ARM. Advanced Micro Bus Architecture (AMBA).. [Online]. Available: http://www.arm.com/products/solutions/AMBAHomePage.html

[22] M. Drinic, D. Kirovski, S. Meguerdichian, and M. Potkonjak, "Latency-guided on-chip bus network design," in *Proc. Int. Conf. Computer Aided Design*, Nov. 2000, pp. 420–423.

[23] M. Sgroi, M. Sheets, A. Mihal, K. Keutzer, S. Malik, J. Rabaey, and A. Sangiovanni-Vincentelli, "Addressing system-on-a-chip interconnect woes through communication-based design," in *Proc. Int. Conf. Design Automation*, Jun. 2001, pp. 667–672.

[24] X. Zhu and S. Malik, "A hierarchical modeling framework for on-chip communication architectures," in *Proc. Int. Conf. Computer Aided Design*, Nov. 2002, pp. 663–671.

[25] N. Thepayasuwan and A. Doboli, "Layout conscious bus architecture synthesis for deep submicron systems on chip," in *Proc. Design Automation and Test in Europe*, Feb. 2004, pp. 10 108–10 115.

Sungchan Kim received the B.S. degrees in material science and engineering and the M.S. degree in computer engineering from Seoul National University, Seoul, Korea, in 1998 and 2000, respectively, where he is currently working toward the Ph.D. degree in electrical engineering and computer science.

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

Chaeseok Im received the B.S. and M.S. degrees in computer engineering and the Ph.D. degrees in electrical engineering and computer science from Seoul National University, Seoul, Korea, in 1997, 1999, and 2004, respectively.

He is presently with Samsung Advanced Institute of Technology, Yongin, Gyeonggi, Korea. His research interests include low power multimedia system design, system level energy estimation, and embedded operating systems.

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 associate professor. His primary research interests are various aspects of embedded system design including Hardware/Software codesign and design methodologies.