Efficient Exploration of Bus-Based System-on-Chip Architectures

Sungchan Kim and Soonhoi Ha, *Member, IEEE*

***Abstract*—Separation between computation and communication in system design allows system designers to explore the communication architecture independently after component selection and mapping decision is made. In this paper, we present an iterative two-step exploration methodology for bus-based on-chip communication architecture for multitask applications. We assume that the memory traces from the processing components are given. The proposed methodology uses a static performance estimation technique extended for multitask applications to reduce the design space quickly and drastically and applies a trace-driven simulation to the reduced set of design candidates for accurate performance estimation. For the case that local memory traffics as well as shared memory traffics are involved in bus contention, memory allocation is considered as an important axis of the design space in our technique. Experimental results show that the proposed methodology achieves significant performance gain by optimizing on-chip communication only, up to almost 100% compared with an initial single shared bus architecture, in both two real-life examples, a four-Channel digital video recorder and an equalizer for OFDM DVB-T receiver.**

***Index Terms*—Communication architecture, design space exploration, memory allocation, multitask, performance estimation.**

I. Introduction

INSATIABLE demand of system performance makes it inevitable to integrate more and more processing elements in a single system-on-chip (SoC) to meet the performance requirement. In addition, with the fast evolution of programmable hardware, such as microprocessors or DSPs, and ever-increasing complexity of its application, a large portion of an application tends to be implemented as multitask software. Such systems usually have complex and diverse on-chip communication traffics among components. As a consequence, on-chip communication design becomes critical for successful SoC designs. In this regard, as a new design methodology, separation between function and architecture and between communication and computation has been recently proposed [1], [23]. Adopting this paradigm, in the proposed design methodology, we model the system behavior as a composition of function blocks and map the function blocks to the processing elements on the target architecture specified separately.

Separation between computation and communication enables system designers to explore the communication architecture independently after component selection and mapping decision is made. Communication architecture decision is performed after a decision is made on which processing elements are used and which function blocks are mapped to where. From the given communication requirements from all processing elements, the design space of communication architecture is explored to find out the optimal one considering the tradeoffs between performance, power, cost, and other design objectives. Since the design space of communication architecture is extremely wide, it is critical to develop an efficient exploration technique, which is the main theme of this paper.

In this paper, we restrict the network architecture to bus since it is still the most popular network [14]–[16], [25]. The design space we explore, however, is still very wide, since it is formed by multiple axes such as the number of buses, bus topology including bus bridges, component allocation, bus arbitration scheme, operation clock frequency, data width, and so on.

Fast and accurate performance estimation is the key to a practical design space exploration methodology. However, speed and accuracy are two conflicting goals of performance estimation. A simulation-based method gives accurate estimation results but pays too heavy a computational cost to be used for exploring the large design space. So, research based on simulation method exploits only a few design axes to confine the design space to be explored. On the other hand, static performance estimation methods do not model dynamic effects accurately enough to determine the optimal architecture. In the proposed technique, however, we utilize the advantages of both approaches by breaking down the exploration procedure into two steps. In the first step, we use a static performance estimation technique to quickly evaluate each candidate design point and prune the design space drastically. The second step uses a trace-driven simulation to accurately evaluate the design points in the reduced design space and to determine the pareto-optimal set of bus architectures.

We assume that processing elements communicate with each other through a shared memory. Each processing element has a single port for both local and shared memory accesses, as usually is the case in real systems. Then, local memory traffics as well as shared memory traffics are also involved in bus contentions: memory allocation is considered as an important axis of the design space in our technique. On the other hand, most previous works have only considered shared memory traffics and have not considered memory allocation separately [5], [8].

Manuscript received July 1, 2005; revised January 8, 2006. This work was supported by the National Research Laboratory Program under Grant M1-0104-00-0015 and the IT Leading Research and Development Support Project funded by Korean MIC.

S. Kim is with the Codesign and Parallel Processing Laboratory, Department of Computer Science and Engineering, Seoul National University, Seoul 151-742, Korea, and also with Samsung Electronics, Yongin, Gyeonggi 446-711 Korea (e-mail: ynwie@iris.snu.ac.kr).

S. Ha is with the Codesign and Parallel processing Laboratory, Department of Computer Science and Engineering, Seoul National University, Seoul 151-742, Korea (e-mail: sha@iris.snu.ac.kr).

Digital Object Identifier 10.1109/TVLSI.2006.878260

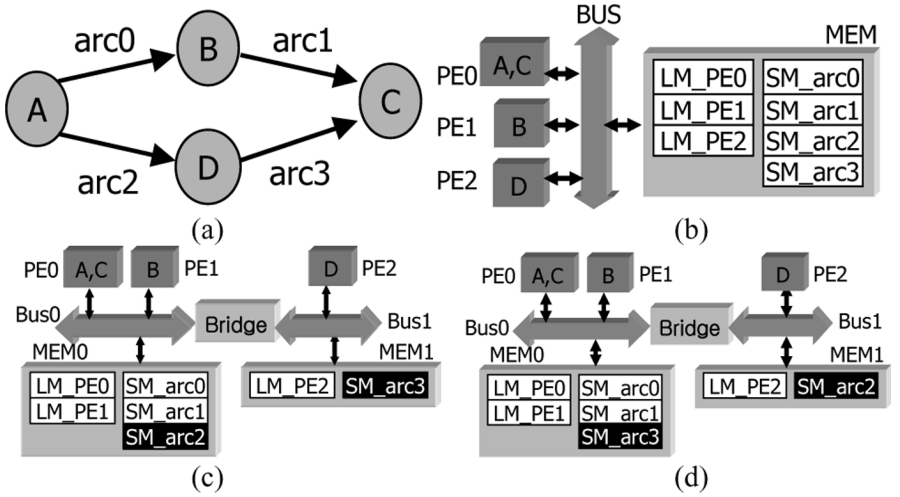


Fig. 1. (a) Behavior specification of an illustrative example, (b) its single shared bus implementation, and (c) and (d) dual-bus implementations with two different mappings of shared memory segments *SM_arc2* and *SM_arc3*.

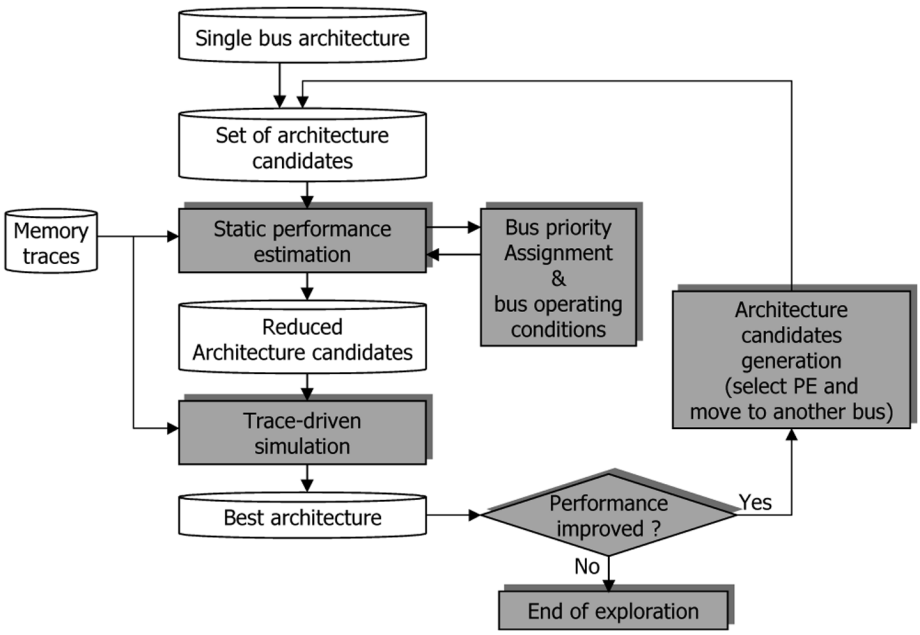


Fig. 2. Proposed design space exploration methodology.

The remainder of this paper is organized as follows. In Section II, we explain the overall procedure of the proposed design space exploration technique with an illustrative example. Section III reviews some related works. In Section IV, we explain the performance estimation technique considering multitask applications. Section V discusses the details of the proposed technique with a preliminary example. Section VI provides the overall structure of the proposed exploration framework. We show some experimental results in Section VII and conclude this paper in Section VIII.

II. Overview of the Proposed Exploration Methodology

For better understanding of the proposed methodology, we use an illustrative example in Fig. 1. Initially, the system behavior is specified as a block diagram of four function blocks. The arcs between function blocks show the data dependency. For example, function blocks B and D can be executed only after function block A is completed. Those four function blocks are mapped to three processing elements: A and C are mapped to processing element *PE0*, B to *PE1*, and D to *PE2*, respectively.

Fig. 1(b) represents a single shared bus implementation. Note that one physical memory component is connected to a bus and it contains seven logical memory segments: three local memory segments and four shared memory segments. Memory segments *LM_PE0*, *LM_PE1*, and *LM_PE2* are local memory segments of *PE0*, *PE1*, and *PE2*, respectively. The arcs between function blocks are implemented as shared memory segments for inter-component communication. For example, *SM_arc0* associated with *arc0* in Fig. 1(a) indicates a shared memory segment for communication between function blocks A and B.

The proposed exploration technique, whose overall structure is shown in Fig. 2, starts the exploration process with this single-bus architecture that becomes the only element in the "set of architecture candidates" initially. A set of memory traces

is one of the inputs to the proposed exploration procedure, which includes both local and shared memory accesses from all processing elements. After the mapping of function blocks to processing elements is completed, the memory traces are obtained using instruction set simulator for each processor core, HDL simulator for ASIC parts, or IP simulators, assuming that memory access overhead is zero. Memory traces are classified into three categories: code memory, data memory, and shared memory. Code and data memories are associated with local memory accesses and shared memory with inter-component communication. For the processor that uses a cache memory, the traces associated with code and data memories represent the memory accesses incurred by cache-miss. Note that the memory trace information is never changed throughout the exploration.

We traverse the design space in an iterative fashion as shown in Fig. 2. The body of the iteration loop consists of three main steps. The purpose of the first step is to quickly explore the design subspace of architecture candidates to build a reduced set of design points to be carefully examined in the next step. With a given set of architecture candidates, we visit all design points by varying the priority assignment of processing elements on each bus and other bus operation conditions. The performance estimation technique used in this step is an extension of [10] to consider multitask applications. We collect the design points with performance difference by less than 10% compared with the best because the proposed static estimation method has less than 10% error bound. The proposed estimation method is based on the queuing model of the system where processing elements are regarded as customers and a bus with its associated memory is regarded as a single server. The service request rate from each processing element is extracted from the memory traces as a function of execution time. Since our method considers the bus contention effect, it gives reasonably accurate estimation results to be used as the first-cut pruning of the design space.

The second step applies trace-driven simulation to the selected design points from the first step. It accurately evaluates the performance of design points in the reduced space and determines the best design point. If the performance of the best design point is not improved from the previous iteration, we exit the exploration loop. Otherwise, we go to the third step and repeat another round of iteration.

The third step generates the next set of architecture candidates. From the architecture of the best design point, we explore the design space incrementally by selecting a processing element and allocating it to a different bus or a new one. Let us go back to the example of Fig. 1. Since the first round starts with one architecture candidate, single-bus architecture, it becomes the input architecture to the third step. Suppose that *PE2* is selected and allocated to a new bus to make a dual-bus system. Since all local memory segments should reside in the same bus as the associated processing element, there are four candidate architectures depending on where to put the shared memory segments associated with *PE2: SM_arc2* and *SM_arc3*. Function blocks A and D use shared memory segment *SM_arc2* so that it may be allocated either to *Bus0* or *Bus1*. However, *SM_arc0* that is accessed by function blocks A and B should remain at *Bus0* since all of its associated processing elements reside in the same bus. Among four candidate architectures, Fig. 1(c) and (d) shows two candidate architectures. In the case 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 way, we can generate 24 candidate architectures for the second round of iteration by moving a processing element into a new bus and considering all possible shared memory segment allocation.

As the iteration goes, we record the best performance numbers as a function of the number of buses to obtain the pareto-optimal design points. If the number of buses increases, the performance tends to increase. We exit the iteration when no performance increase is obtained from the previous iteration.

The proposed technique does not explore the entire design space but it is a greedy heuristic to prune the design space aggressively since we select only the best architecture at the end of iteration. If we select multiple ones, we may explore the wider set of design points with longer execution time.

III. Related Work and Our Contribution

Some researchers have considered communication architecture selection simultaneously during the mapping step. Since the communication overhead is needed for the mapping decision, static estimation of communication architecture has been investigated. A technique was proposed to estimate the communication delay using the worst-case response analysis of real-time scheduling [1]. Knudsen and Madsen estimated the communication overhead on a point-to-point channel taking into account the data transfer rate variation depending on the protocol, configuration, and different operating frequencies of components [13]. Nandi and Marculescu proposed a performance measure technique based on continuous-time Markov processes [11]. However, these techniques did not model the dynamic effects such as bus contention and explored only a limited configuration space.

For exploration of communication architectures, simulation-based estimation is widely adopted in many commercial tools and academic researches at various transaction levels [4], [12], [22]. A simulation-based method gives accurate estimation results but pays too heavy a computational cost to be used for exploring the large design space. So, previous research based on this method has exploited only a few design axes to confine the design space to be explored. To overcome this difficulty, a hybrid approach between a static estimation and a simulation approach has been developed by Lahiri *et al.* [3]. They used 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 applied some static analysis to the memory traces to reduce the time complexity of trace-driven simulation.

Since the design space is extremely huge, most previous works focused on a small number of design axes. In Gong *et al.*'s work [6], system specification refinement onto four fixed communication architecture templates was addressed to optimize performance. Gasteier *et al.* proposed a bus topology synthesis technique at high level using the port constraints of components and considering only static information such as bit widths, the amount of data transfer, and so on [7]. Meeuwen *et al.* presented a technique for cost-efficient interconnect architecture exploration by time-multiplexing the data transfers over

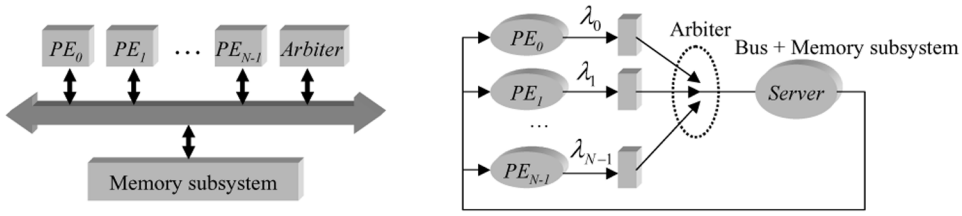


Fig. 3. Queuing model of a single bus.

a number of shared buses assuming distributed memory systems under predetermined memory allocation [8]. Meftali *et al.* found performance-optimal shared memory allocation considering area of communication channel and memory subsystem using integer linear programming (ILP) in a point-to-point communication architecture [9]. Lahiri *et al.* proposed an exploration technique optimizing the component mapping to bus and the bus protocol such as DMA block transfer size and bus priority assignment for a given bus topology [5]. Thepayasuwan and Doboli proposed a bus architecture synthesis technique that minimizes the cost considering bus topology, communication conflicts, and bus utilization using a simulated annealing [20]. Srinivasan *et al.* developed a technique that performs both bus partitioning and bus frequency assignment simultaneously to optimize power consumption and performance using a genetic algorithm [21]. A technique using the profiled statistics of communication traffics between cores to determine core-to-bus assignment for a given application was proposed by Drinic *et al.* [24].

Compared with these related works, the main contribution of the proposed technique is that we explore the larger design space by the two-step design space exploration, considering multiple design axes such as the number of buses, bus topology, component allocation, priority assignment, and other bus operating conditions. Since the proposed exploration technique is extensible, more design axes can easily be added. Our work also extends the previous works in two ways. We consider local memory accesses as well as shared memory accesses. Previous works on architecture exploration are mostly concerned with only communication requirements between processing elements ignoring the local memory accesses. In case local memory accesses are also involved in bus contention, they need to be considered. Another extension is our static performance analysis using the queuing model capable of considering multitask applications.

IV. Performance Estimation of Multitask Applications

Here, we discuss the extension of our previous performance estimation technique to multitask applications. We first summarize the performance estimation technique of single task proposed in [10]. The technique is based on the queuing model of the target bus architecture, where processing elements are customers while a bus and associated memory subsystem correspond to a single server. The model aims at estimating an average wait time for bus grant of each processing element by construction of a steady-state transition diagram.

Fig. 3 shows the queuing model of a single bus architecture. There are N processing elements $(PE_0, PE_1, \ldots, PE_{N-1})$ competing for the use of a bus. It is assumed that the bus arbitration is based on the fixed priorities of processing elements. PE_0 is assigned the highest priority. The bus access is assumed nonpreemptive.

λ_i denotes the rate at which the processing element PE_i issues memory requests assuming an ideal condition of zero memory access overhead. For function block fb on a processing element, we first compute the bus request rate $ar(fb)$, the schedule length $sl(fb)$, and the average bus access time $ba(fb)$ from the memory traces' information and scheduling result, assuming ideal but unrealistic bus conditions: no waits for bus grant and the access time of one cycle for unit data transfer from/to memory. The bus request rate λ_i is a ratio of bus access counts $bc(fb)$ over $sl(fb)$ so that it varies according to the function block running on PE_i.

If the execution time is lengthened due to bus contention, the effective arrival rate of requests becomes smaller than λ_i. We denote 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, and 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 [28] 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-hand side of (1). To obtain this, we use a state transition diagram and its steady-state probability. As a result, dynamic bus conflicts can be predicted accurately. We omit the detailed explanation on the queuing model; refer to [10] for further details.

Fig. 4(a) shows an example of static schedule of a task that consists of four function blocks, assuming ideal bus conditions. 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. 4(a). Through the queuing analysis, we

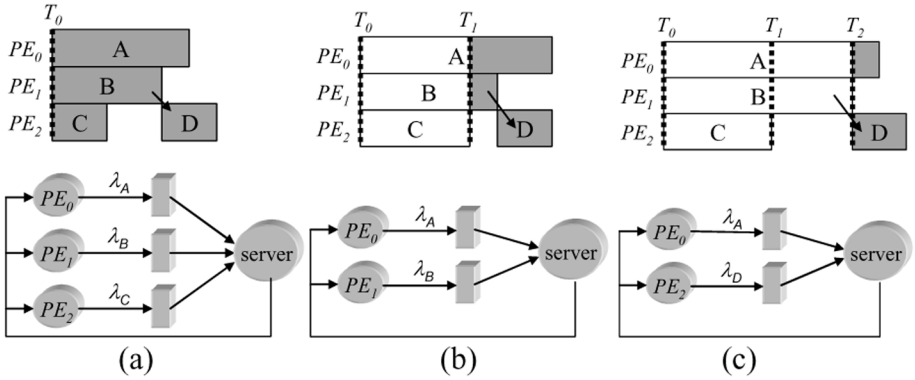


Fig. 4. (a) Example schedule of PE_0, PE_1, and PE_2 and the corresponding queuing model. (b) New queuing model after function block C is finished at T_1. (c) Another queuing model after function block B is finished at T_2.

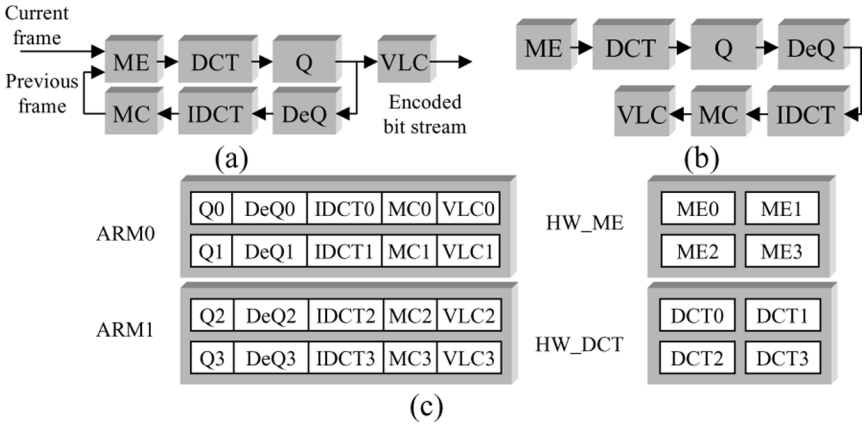


Fig. 5. (a) Specification and (b) schedule of H.263 encoder and (c) the mapping of function blocks to processing elements for a four-channel DVR.

estimate the waiting time due to bus contentions to obtain the estimated schedule extension as illustrated in Fig. 4(b) until any function block completes its execution. In this example, function block C is assumed to be finished first at T_1. We consider the function blocks that are concurrently executable after that time. After T_1, two function blocks A and B continue executions until either one finishes next. Thus, we construct the associated queuing model with PE_0 and PE_1 as shown in Fig. 4(b). The shaded regions indicate the remaining parts of the static schedule to be estimated by the queuing analysis. Such evaluation process is repeated until all of the function blocks are examined.

Now we consider multitask applications. For simplicity, but with little loss of generality, we make the following assumptions: all tasks are independent, any preemptable task scheduling policy can be used, and the scheduling overhead is negligible. In case tasks are interdependent, we still consider the set of independent tasks that run concurrently with the function block of interest for static analysis.

We explain the proposed technique using a four-channel digital video recorder (DVR) example throughout this paper. The DVR receives the raw bit streams from external four 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. 5(a) and (b) shows the specification and the schedule of an H.263 encoder, respectively, while Fig. 5(c) represents a mapping example of function blocks

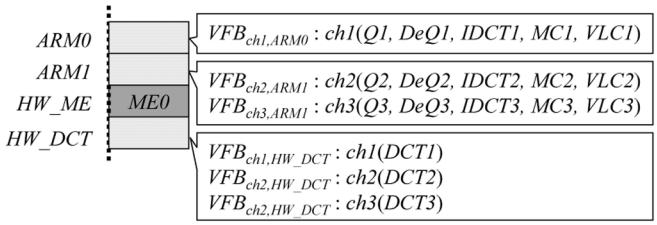


Fig. 6. Function blocks of *ch1*, *ch2*, and *ch3* on *ARM0*, *ARM1*, and *HW_DCT* that can be executed simultaneously with *ME0*.

onto processing elements. Each ARM processor takes charge of running two tasks, respectively. A task is mapped to one ARM processor except for the function blocks *ME* and *DCT*, which are mapped to the dedicated hardware blocks *HE_ME* and *HW_DCT*, respectively. Memory traces are obtained from encoding of a P-frame in QCIF-format bit stream.

Suppose we want to estimate the execution time of *ME0* of task *ch0* mapped to *HW_ME*. To build the queuing model as depicted in Fig. 6, the function blocks being concurrently executed on *ARM0*, *ARM1*, and *HW_DCT* with *ME0* should be selected. In *ch0*, no function blocks are concurrently executable with *ME0* due to the execution dependency. Therefore, only *ch1* can be simultaneously executable with *ME0* in *ARM0* since *ME0* of task *ch0* is executed in *HW_ME*. However, it cannot be statically determined which function block of *ch1* is concurrently executable with *ME0*. Moreover, any function block of two tasks *ch2* and

ch3 can be executed in *ARM1*. If we enumerate all possible combinations of the function blocks concurrently executable with *ME0*, $5 \times \binom{2}{1} \times 5 \times 3$ or 150 queuing models should be investigated, which is impractical for fast design space exploration. Thus, we propose a heuristic approach.

As explained in the beginning of this section, the required parameters for constructing the queuing model are the bus request rate and the average bus access time. To model the bus contentions due to other tasks, we define a virtual function block $VFB_{\tau,pe}$ for task τ on processing element pe, which has an approximated bus request rate $ar(VFB_{\tau,pe})$ and a bus access time $ba(VFB_{\tau,pe})$ while *ME0* is executed. $ar(VFB_{ch1,ARM0})$ and $ba(VFB_{ch1,ARM0})$ of $VFB_{ch1,ARM0}$ are computed as follows:

$$ar(VFB_{ch1,ARMO}) = \frac{\sum bc(fb)}{sl(ch1)} \tag{3}$$

$$ba(VFB_{ch1,ARM0}) = \frac{\sum \{ba(fb) \cdot bc(fb)\}}{\sum bc(fb)} \tag{4}$$

where $fb \in$ {the function blocks of ch1 mapped to $ARM0$} and *sl*(*ch1*) is the schedule length of the task *ch1*. In other words, $ar(VFB_{\mathrm{ch1,ARM0}})$ and $ba(VFB_{\tau,pe})$ mean an average request rate and an average bus access time of the function blocks of *ch1* that are concurrently executable with *ME0* in *ARM0*, respectively. In *ARM1*, two virtual function blocks $VFB_{\mathrm{ch2,ARM1}}$ and $VFB_{\mathrm{ch3,ARM1}}$, are defined. Then, the virtual function block with higher bus request rate is selected. We also assume that the schedule length of all virtual function blocks is infinite to make them longer than *ME0*.

For more general formulation, we define two terms $T(fb)$ and $P(fb)$ that are the task including function block fb and the processing element executing 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 function block $fb*$ is executed on processing element $PE*$, i.e., $P(fb*) = pe*$. In order to build the queuing system of Fig. 4, $\psi_{fb*,pe}$ is defined as the virtual function block of processing element pe, which is assumed to be running concurrently with function block $fb*$. Therefore, the bus request rate $ar(\psi_{fb*,pe})$ of $\psi_{fb*,pe}$ becomes

$$ar(\psi_{fb*,pe}) = \max_{\forall\, \tau, \tau \neq \tau*} \left\{ \frac{\sum_{fb_i} bc(fb_i)}{sl(\tau)} \right\} \tag{5}$$

where $fb_i \in \{fb \mid T(fb) = \tau, fb//fb*, P(fb) = pe, Pe \neq Pe*\}$ and $\tau* = T(fb*)$. Thus, among all tasks running on pe, we choose the worst case that has the highest bus request rates. If task τ is selected to build the virtual function block of pe by (5), the average bus access time $ba(\psi_{fb*,pe})$ of virtual function block ψ_{fb*,p_e} in ideal bus conditions is

$$ba(\psi_{fb*,pe}) = \frac{\sum_{fb_i} \{ba(fb_i) \cdot bc(fb_i)\}}{\sum_{fb_i} bc(fb_i)} \tag{6}$$

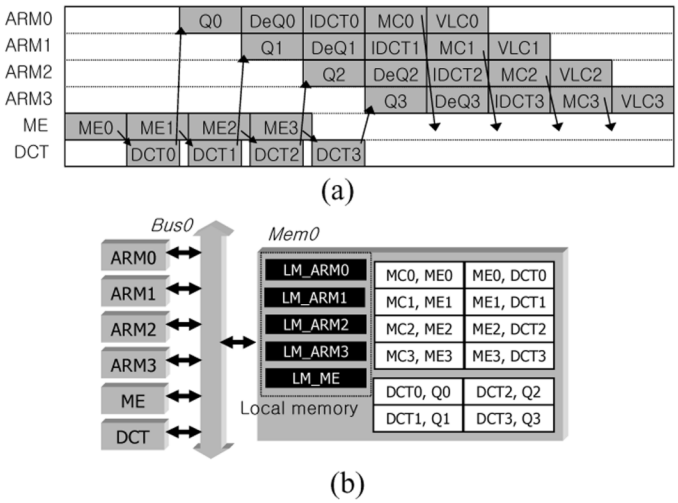


Fig. 7. (a) Initial schedule of four-channel DVR and (b) its single-bus implementation.

where $fb_i \in \{fb \mid T(fb) = \tau, P(fb) = pe, Pe \neq Pe*\}$. With those queuing parameters of the processing element, the queuing system is constructed to estimate the average wait time for bus grant of function block $fb*$.

V. Generation of Communication Architecture Candidates

Here, we explain how the proposed technique explores the design space of a four-channel DVR example in Fig. 5.

Suppose that each H.263 encoder is mapped to a separate ARM processor except all *ME* and *DCT* blocks: they are mapped to a motion estimation (*ME*) hardware component and a discrete cosine transform (*DCT*) hardware component, respectively. Thus, four H.263 encoders share two hardware components. Fig. 7(a) and (b) shows the scheduling and mapping result of a four-channel DVR and its single-bus implementation, respectively. This example system has 14 memory segments: five local memory segments and 12 shared memory segments. In Fig. 7(b), for instance, shared memory segment "*MC0, ME0*" is associated with communication between function blocks *MC0* and *ME0*.

A. Bus Topology Exploration

Performance improvement of communication architecture can be achieved by scattering communication traffics into multiple buses to reduce bus contention and to maximize concurrency. For this purpose, we select a processing element and allocate it to a new bus or to another existing bus. Suppose that we select *ARM0* in the single-bus architecture of Fig. 7. The way of changing the allocation of *ARM0* is only to create a new bus, *Bus1*. Then, its associated shared memory segments "*MC0, ME0*" and "*DCT0, Q0*" can be allocated to either *Bus0* or *Bus1* to generate four architecture candidates as shown in Fig. 8. It is important to note that a bus bridge is introduced between two buses for inter-bus communication.

Further communication traffic reduction can be obtained by removing local memory accesses of each processing element from shared buses, as illustrated in Fig. 9. Processor *ARM0*

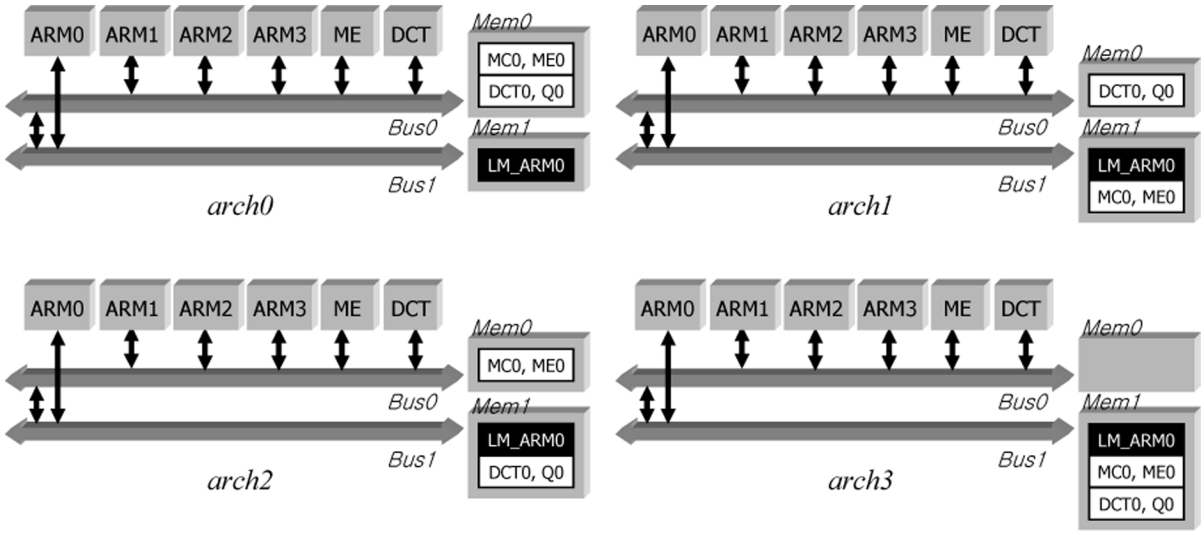


Fig. 8. Creating a new bus for processing element *ARM0* and allocating its associated shared memory segments.

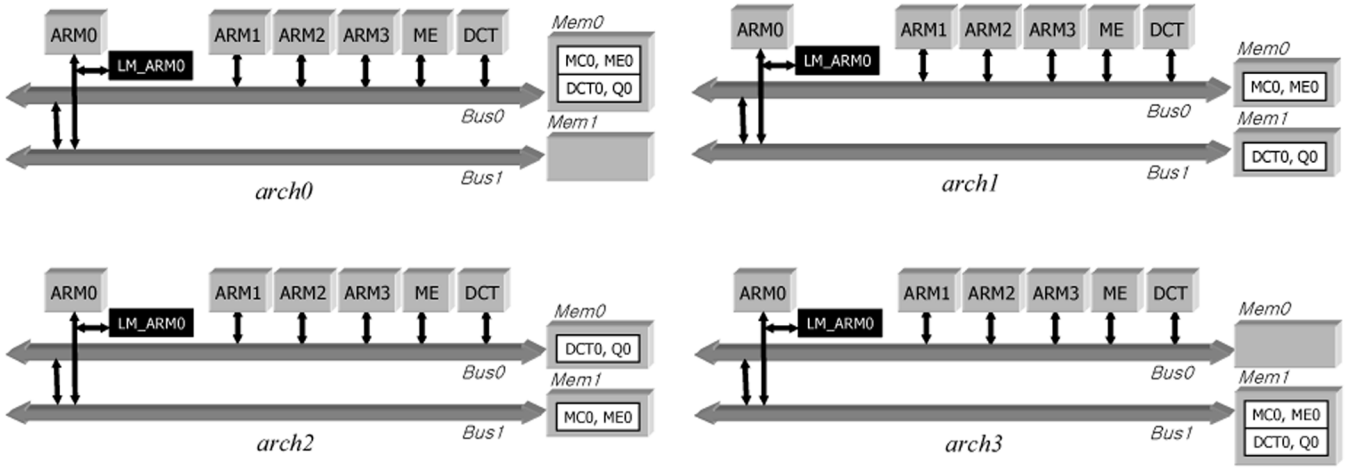


Fig. 9. If the local memory segment of processor *ARM0* is separated from shared bus *Bus1*, communication traffics incurred by local memory accesses can be removed from *Bus1*, which may lead to further performance improvement.

can access its local memory segment *LM_ARM0* without contending for the use of *Bus1*. Since the local memory segment in Fig. 8 is not a cache, the requests from a processing element go out to either the bus or the local memory according to its destination address. A drawback of such separation is area overhead caused by adding a local bus to each processing element. However, it may be desirable for power saving due to the use of more memory components of smaller size as well as reduced bus-switching activities [26], [27]. We leave it as future work to consider power consumption as another design objective.

Further communication traffic reduction can be obtained by removing local memory accesses of each processing element from shared buses as illustrated in Fig. 9. Processor ARM0 can access its local memory segment LM_ARM0 without contending for the use of Bus1. Since the local memory segment in Fig. 8 is not a cache, the requests from a processing element go out to either the bus or the local memory according to its destination address. A drawback of such separation is area overhead caused by adding a local bus to each processing element. It, however, may be desirable for power saving due to the use of more memory components of smaller size as well as reduced bus-switching activities [26], [27]. We leave it as a future work to consider power consumption as another design objective.

TABLE I
NUMBER OF GENERATED ARCHITECTURES FROM THE SINGLE-BUS ARCHITECTURE OF FOUR-CHANNEL DVR BY CHANGING ALLOCATION OF SHARED MEMORY SEGMENTS

Processing element being moved	ARM0	ARM1	ARM2	ARM3	ME	DCT
Number of shared memory segments	2	2	2	2	8	8
Number of generated architectures	2^2	2^2	2^2	2^2	2^8	2^8
Total	528					

Table I shows the size of design space by moving each processing element from the single-bus architecture in Fig. 7(b). In total, 528 architecture candidates are generated. The same number of architecture candidates is also generated for the

architectures that use local buses for local memory segments. Consequently, more than one thousand architecture candidates should be investigated by examining the bus topology only.

B. Bus Parameterization

Once the bus topology and memory allocation are determined, bus protocol of each architecture candidate should be configured. Bus protocol includes priority assignment, operation clock frequency, bus data-width, and so on. Of these, priority assignment should be treated with care since it may cause significant performance variation, as observed in [5] and [18]. Although an exhaustive search method guarantees an optimal result, it is prohibitively expensive. For example, the first architecture, *arch0*, of Fig. 8 has six bus masters in *Bus0* and two bus masters in *Bus1* including bus bridges. Since 6! or 1440 priority assignments for *Bus0* and 2! assignments for *Bus1* are possible, 1440×2 assignments should be investigated in an exhaustive search method to get the optimal priority assignment. To overcome this difficulty, we devised a priority assignment heuristic where a higher priority is bestowed to the processing element with more memory accesses and to more critical processing element.

The amount of data transfer per unit time $BW(fb_i)$, i.e., bandwidth, represents the memory access characteristics of function block fb_i. The criticality $C(fb_i)$ of function block fb_i is the sum of the schedule length of function blocks on the longest execution path starting from block fb_i. The bandwidth and the criticality of a function block are computed from the memory traces and the function block scheduling, respectively. In our heuristic, we define the rank $R(fb_i)$ of function block fb_i as the product of criticality $C(fb_i)$ and bandwidth $BW(fb_i)$. Also, the rank $R(PE)$ of processing element PE is the sum of the ranks of function blocks that are executed in PE. Thus, we get following formula:

$$R(PE) = \sum_{fb_i \in FB_{PE}} R(fb_i) = \sum_{fb_i \in FB_{PE}} \{BW(fb_i) \cdot C(fb_i)\} \quad (7)$$

where FB_{PE} is the set of function blocks mapped on PE. The higher the rank of a processing element is, the higher priority the processing element is assigned. After this initial assignment, we perform a simple annealing process by swapping the priorities of two processing elements on the same bus.

When the priority assignment of a bus is investigated, assignments of the other buses are assumed fixed. For instance, in *arch0* of Fig. 8, we start by varying the priorities of processing elements in *Bus0* first. The performance estimation proposed in the previous section is applied to every combination by swapped priorities between two processing elements. Therefore, $\binom{6}{2}$ or 30 architecture candidates are investigated when considering *Bus0* and the best result is selected as the priority assignment of *Bus0*. Then, we move to *Bus1* and repeat the same procedure to find a more improved solution, which results in $\binom{3}{2}$ or six priority assignments. Therefore, the total number of assignments to be explored by the heuristic amounts to 36. The proposed assignment heuristic shows remarkable results compared with an exhaustive search method, while it explores significantly reduced design space, by 80 times in this example. We validate its efficiency by experimental results in Section VII.

```
1:  Select_Architecture (Initial_Arch, Sched, Mem_Trace)
2:    arch_list→Initialize(Initial_Arch)
3:    while (true) do
4:      Num_Qualified_1st = 0
5:      // 1st pruning step
6:      for each arch_i ∈ arch_list, i=1,2,...,N_1st do
7:        Bus_Protocol_Synthesis(arch_i)
8:        Do_Performance_Estimation(arch_i, Sched, Mem_Trace)
9:      end for
10:     Best_Exe_Time = Find_Best_Exe_Time(arch_list)
11:     for each arch_i ∈ arch_list , i=1,2,...,N_1st do
12:       if ((arch_i→Exe_Time - Best_Exe_Time)
13:             / Best_Exe_Time > ESTIMATION_ERROR) then
14:         arch_list→Delete(arch_i)
15:       else
16:         Num_Qualified_1st = Num_Qualified_1st + 1
17:       end if
18:     end for
19:     if (Num_Qualified_1st > MAX_ARCH) then
20:       for each arch_i ∈ arch_list , i=1,2,..., N_1st do
21:         if (i > Num_Qualified_1st) then
22:           arch_list→Delete(arch_i)
23:         end if
24:       end for
25:     end if
26:     // 2nd pruning step
27:     Prev_Seed_Arch = Curr_Seed_Arch
28:     for each arch_i ∈ arch_list , i=1,2,...,N_2nd do
29:       Do_Trace_Driven_Simulation(arch_i, Sched, Mem_Trace)
30:       if (Curr_Seed_Arch→Exe_Time < arch_i→Exe_Time) then
31:         Curr_Seed_Arch = arch_i
32:       end if
33:     end for
34:     // check the termination condition
35:     if ((Curr_Seed_Arch→Exe_Time >= Prev_Seed_Arch→Exe_Time) or
36:       (Curr_Seed_Arch→Num_Bus == Curr_Seed_Arch→Num_Pe)) then
37:       Quit_Exploration( );
38:     end if
39:     Generate_Architecture_Candidates(arch_list, Curr_Seed_Arch)
40:   end while
41: end Select_Architecture
```

Fig. 10. Proposed exploration flow.

Bus clock frequency and bus data-width depend on the memory used. For brevity, in this paper, we assume that all buses in an architecture candidate are synchronized with a single global clock and its frequency is set to be the reciprocal of memory access time for one word. Bus data-width follows the data-width of memory.

VI. Overall Structure of the Exploration Framework

Fig. 10 summarizes the main procedure of the proposed technique: **Select_Architecture**. This procedure requires three inputs: the initial architecture *Initial_Arch* to begin the exploration, the schedule information of system specification *Sched*, and the memory traces *Mem_Trace* of processing elements. The **while** statement of line 3 defines the main iteration loop of exploration.

Select_Architecture consists of three parts. The first part is the first architecture-pruning step from line 5. Initially, the set of architecture candidate contains only one element, *Initial_Arch*. In the first **for** loop, from line 6 to line 9, the diverse

priority assignments selected from the proposed priority assignment heuristic are assessed by the proposed static estimation method, and we obtain the best performance of each architecture candidate.

Then, after the best performance values of all architecture candidates are sorted in an ascending order, the architecture that has the shortest execution time *Best_Exe_Time* is chosen. Since our static estimation method is observed to have 10% error bound through preliminary examples, the architecture candidates that have the estimated performance differed from *Best_Exe_Time* by less than 10% may have actually better performance than *Best_Exe_Time*. Thus, this error range is used to reduce the design space: the parameter *ESTIMATION_ERROR* is set to 0.1. In the **for** loop from lines 11 to 18, if the performance difference of an architecture candidate from *Best_Exe_Time* is greater than *ESTIMATION_ERROR*, it is pruned from the design space. Note that the more accurate the static estimation technique is, the narrower the design space becomes that should be investigated more precisely in the second pruning step.

In case the reduced design space is still too large, we may want to restrict the maximum number of architecture candidates to be explored in the second step. Therefore, we define the *MAX_ARCH* parameter and enforce that at most as many as *MAX_ARCH* architecture candidates are left in the reduced design space (lines 19–25).

The second part of the procedure applies trace-driven simulation to the selected architecture candidates from the first step. We use an in-house cycle-accurate trace-driven simulator at this step. Since the estimated performance from the trace-driven simulation is very accurate, we compare the performances of all candidate architectures and choose the best architecture. Then, the performance of the best architecture is compared with that of the previous iteration. If no performance improvement is achieved from the current iteration or the number of shared buses reaches the number of processing elements, we exit the iteration loop and terminate the procedure.

The last part of procedure **Select_Architecture** is to generate the architecture candidate incrementally from the best architecture chosen from the second part (line 39). How to generate the architecture candidate is already explained in the previous section. When we estimate the performance, we record the best performance value for each number of buses used to obtain the pareto-optimal set of bus architectures.

VII. Experimental Results

This section provides the experimental results on the static estimation technique for multitask extension, on the priority assignment heuristic, and on the proposed two-phase exploration methodology. All experiments were conducted on a Xeon 2.8-GHz workstation running Linux.

A. Validation of the Proposed Performance Estimation Technique

We compared the static estimation result from the proposed multitask extension of the previous queuing analysis for the four-channel DVR system example with a trace-driven simulation. Trace-driven simulation used in this experiment 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 [19]. However, no scheduling overhead is considered in the simulator.

TABLE II
Results of the Exploration fora Four-Channel DVR

Number of buses	Number of architectures	Estimated execution time
1	1	24869021
2	2208	22297455
3	1072	22502381
4	512	22486609
Total	3793	
Estimation/arch in the1st step (sec)	0.83	

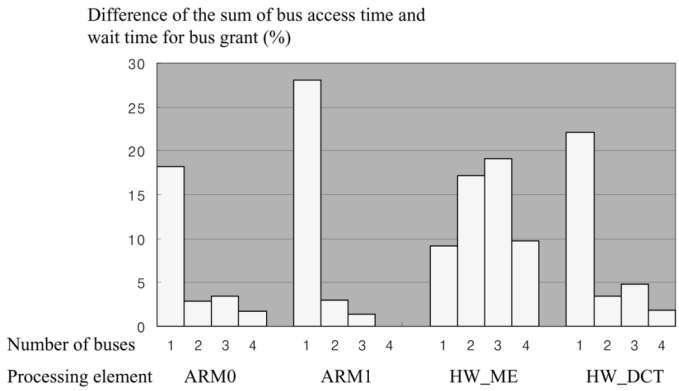


Fig. 11. Difference between the sum of bus access time and wait time for bus grant between the proposed static estimation and the trace driven simulation for each processing element on a four-channel DVR.

Table II represents the results after architecture exploration for a four-channel DVR until no more performance gain is obtained. In Table II, the second column "*Number of architectures*" shows the number of generated architecture candidates according to the associated number of buses during the exploration. The last row indicates the average elapse of estimating an architecture candidate in the first exploration step. It takes no less than one second, which shows the effectiveness of the proposed technique. In the third column "*Estimated execution time*," the best performance obtained from the trace-driven simulation is recorded in bus clock cycles. No performance gain is obtained with more than two buses, since the overhead of crossing bus bridges tends to exceed the benefit of scattering communication traffics by splitting a bus.

Fig. 11 shows the difference of the sum of bus access time and waiting time for bus grant between the proposed static estimation and trace-driven simulation. Each bar corresponds to the maximum difference over the explored architectures with the same number of buses for a four-channel DVR. The estimation error compared with the trace-driven simulation is about 28% in the worst case. Since the ratio of bus access time over the entire execution does not exceed 30%, the estimated error on the entire execution becomes less than 10%. Furthermore, the average estimation error is around about 6%. Through the experiments, we verify that the proposed estimation technique

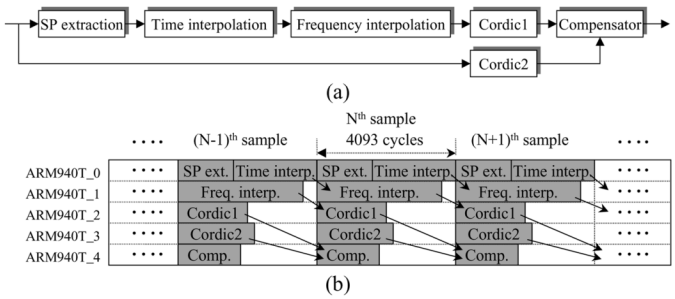


Fig. 12. (a) Specification of the equalizer for OFDM DVB-T receiver and (b) its schedule.

TABLE III
EFFICIENCY OF THE PRIORITY ASSIGNMENT HEURISTIC COMPARED WITH AN EXHAUSTIVE METHOD: EQUALIZER FOR THE OFDM DVB-T RECEIVER

Target architecture	Single bus		Dual Bus	
Assignment method	Exhaustive	Heuristic	Exhaustive	Heuristic
# of architectures	1		30	
# of total assignments per architecture	120	10	144 ~ 240	9 ~ 11
# of average assignments per architecture	120	10	195	10
Performance	Variation	Initial / Tuning	Variation	Initial / Tuning
	1 ~ 1.32	1.18 / 1.006	1 ~ 1.21	1.08 / 1.006

can be used successfully for reducing the design space for multitask applications.

B. Validation of the Priority Assignment Heuristic

We compared the efficiency of the proposed priority assignment heuristic with the exhaustive assignment for architecture candidates during exploration of two examples: the equalizer subsystem of an OFDM DVB-T receiver and a four-channel DVR system, starting from the single-bus architecture. In a DVB-T receiver, the equalizer is used for correcting the amplitude distortion of received signals [17]. Function blocks of the equalizer are mapped onto five ARM940T processors and are scheduled in a pipelined fashion to make all processors run concurrently, as shown in Fig. 12(b). Due to an excessively long run time of the exhaustive search, we only considered single-bus and dual-bus implementations for performance comparison. Comparison results are summarized in Tables III and IV for each example, respectively.

For each bus topology, the number of investigated combinations of priority assignment by exhaustive search and the heuristic are given in the rows "*# of total assignments per architecture*" and "*# of average assignments per architectures*" in their total and average, respectively. Comparing with the exhaustive search, the proposed heuristic assignment reduces

TABLE IV
EFFICIENCY OF THE PRIORITY ASSIGNMENT HEURISTIC COMPARED WITH AN EXHAUSTIVE METHOD: FOUR-CHANNEL DVR

Target architecture	Single bus		Dual Bus	
Assignment method	Exhaustive	Heuristic	Exhaustive	Heuristic
# of architectures	1		183	
# of total assignments per architecture	720	15	720	15
# of average assignments per architecture	720	15	720	15
Performance	Variation	Initial / Tuning	Variation	Initial / Tuning
	1 ~ 1.22	1.19 / 1.000	1 ~ 1.26	1.08 / 1.002

the search space significantly by about 12 times in a single-bus implementation and about 19 times in dual-bus implementations. We evaluated the performance of all combinations of priority assignment by exhaustive method and then recorded the normalized values with respect to the best whose performance is set to 1. The smaller the value is, the better the performance is. It should be noticed that wrong priority assignment leads to 30% performance degradation in the worst case. It confirms the importance of optimal priority assignment.

The column "*Initial/Tuning*" reports the results by the proposed heuristic. The first value is obtained from the initial assignment while the second one is after the annealing process. As shown in the tables, initial assignment does not always guarantee acceptable results. Even though the quality of initial assignment only is not good, it can be elevated close to the optimum by the annealing process. In both examples, the heuristic results are deviated from the optimum only by 1% at most for various bus architectures. In the case of singe-bus implementation of the four-channel DVR, the optimum was found by the heuristic. It shows that the proposed heuristic is effective to find an optimized priority assignment as well as to reduce the search space drastically.

C. Validation of the Proposed Exploration Methodology

The proposed two-phase exploration technique is applied to the previous examples. The maximum number of architectures to be evaluated in the second pruning step, *MAX_ARCH* in Fig. 10, was fixed to 20. Table V represents the results of exploration for each example system. We do not separate local memory accesses to the local buses in this experiment. Exploration considering local buses will be discussed in the next experiment to investigate additional performance improvement due to local buses. In each set of Table V, the first column "*# of arch*" shows the number of generated architecture candidates having the associated number of buses during whole exploration. The number of processing elements of a system is equal to the maximum number of buses that an architecture candidate may have.

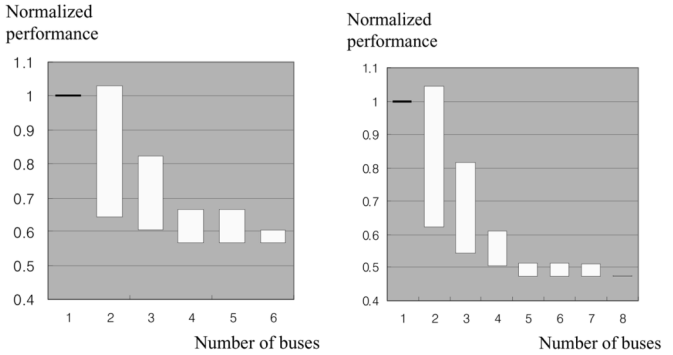
TABLE V
RESULTS OF EXPLORATION FOR THE EXAMPLE SYSTEMS

Application	4-Channel DVR		Equalizer for DVB-T	
Number of shared memory segments	14		4	
Number of buses	Number of architectures	Speed up	Number of architectures	Speed up
1	11	1	7	1
2	5912	1.5511	167	1.5871
3	3753	1.5511	107	1.5871
4	2119	1.7651	68	1.7661
5	1058	1.7686	12	1.7671
6	260	1.7689	-	-
Total	13112		360	
Pruning ratio (%)	99.6		94.4	
Estimation time in the 1st step (sec)	0.67		0.06	
CPU time (sec)	12063		24	

The column "*Speed up*" shows the performance improvement of the best architecture among the architecture candidates compared with the initial single-bus architecture. Performance improvement tends to be saturated near the end of the exploration. It is noteworthy that the maximum performance of example systems is about 50% to 100% better than that of the single-bus architecture. Since such improvement comes from optimization of only communication architecture and memory allocation, it confirms the usefulness of the proposed technique.

The four rows from the bottom represent the number of total architecture candidates explored, the architecture pruning ratio by the static performance estimation, the average time taken for the static performance estimation of an architecture candidate, and the total elapsed time for the exploration, respectively. In the case of the four-channel DVR, a pruning ratio is close to 100%. The entire set of architecture candidates includes those by priority assignments as well as by the move of processing elements and shared memories. As reported in the second row from the bottom, each architecture candidate is evaluated rapidly within less than one second in average. The total execution time of the last row includes trace-driven simulation.

The performance variation of each system according to the number of buses is shown in Fig. 13. For all systems, dual-bus system has the widest performance variation meaning that wrong mapping of processing elements or memory allocation could lead to significant performance degradation. For example, in the four-channel DVR and System3, the worst performance of dual-bus architecture is even inferior to single-bus architecture. However, as more buses are used, the variation becomes smaller since the concurrency of memory accesses is fairly exploited by multiple buses enough to compensate for performance degradation due to wrong mapping of processing elements and memory allocation. If we take the best performance for each number of buses, we obtain the pareto-optimal set of bus architectures.



(a) 4-Channel DVR (b) Equalizer for DVB-T receiver

Fig. 13. Performance variation of the example systems during exploration, varying the number of buses. In each graph, horizontal and vertical axes represent the number of buses used and the normalized execution time, respectively. (a) Four-channel DVR. (b) Equalizer for DVB-T receiver.

TABLE VI
PERFORMANCE IMPROVEMENT OBTAINED BY CONSIDERING LOCAL BUS EXPLORATION

Applications	Equalizer for OFDM DVB-T Receiver	4-Channel DVR
Initial single bus	1.00 (96,478 cycles)	1.00 (21,372,362 cycles)
Not considering local bus	0.67 (64,704 cycles)	0.57 (12,086,437 cycles)
Considering local bus	0.57 (55,012 cycles)	0.48 (10,227,737 cycles)

As the last experiment, we examined how much performance improvement can be obtained by using dedicated local buses for local memory access, as discussed in Section V-A. The same environment and configurations of the previous experiment are used again for equalizer and DVR examples. Contrary to the previous experiment, local buses of processing elements are explored to get further performance improvement. Similar tendencies could be observed as shown in Table V and Fig. 13. Now, we focus on how much performance improvement is obtained and summarize the results in Table VI. For each example, performance values for three types of architecture are reported: Initial single bus architecture, the best architecture without local buses, and the best one with the local buses. Performance values are provided in both normalized ones and bus clock cycles. The performance of initial single bus implementation is set to 1. The performance improvement with local buses is about 100%, i.e., it becomes two times faster than initial single-bus architectures for both examples. It is about 20% better than the architecture without local buses.

VIII. CONCLUSION

In this paper, we have presented an iterative two-step exploration technique of bus-based on-chip communication architectures and memory allocation. At each iteration, the first step quickly reduces the large design space drastically by using an efficient static performance estimation method based on a queuing model. In the second step, the reduced design space is explored using a trace-driven simulation to choose the best architecture candidate. Experimental results with two examples,

a four-channel DVR and the equalizer subsystem for OFDM DVB-T receiver, and three randomly generated examples validated the efficiency and the viability of the proposed technique to explore the wide design space.

The main contribution of the proposed technique is that we explored the larger design space by the two-step design space exploration, considering multiple design axes such as the number of buses, bus topology, component allocation, priority assignment, and other bus operating conditions. Since the proposed exploration technique is extensible, more design axes can easily be added. We also extended the previous works to consider local memory accesses and multitask applications in the proposed static performance estimation.

The main overhead of the proposed methodology is building of the trace-driven simulator and a new queuing model for static performance analysis for a new bus standard of interest. Even though only the performance metric has been investigated in this paper, the proposed methodology is extensible to consider other metrics such as power consumption, which is currently under development. Another future work is the extension of the proposed methodology to off-chip system, i.e., board-level system, and to network-on-chip architecture.

ACKNOWLEDGMENT

The authors would like to acknowledge the useful comments and suggestions of the anonymous reviewers who helped improve the quality of this paper. The ICT and ISRC at Seoul National University and IDEC 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. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 19, no. 12, pp. 1523–1543, Dec. 2000.

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

[3] K. Lahiri, A. Raghunathan, and S. Dey, "System-level performance analysis for designing system-on-chip communication architecture," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 20, no. 6, pp. 768–783, Jun. 2001.

[4] P. Lieverse, P. van der Wolf, K. Vissers, and E. Deprettere, "A methodology for architecture exploration of heterogeneous signal processing systems," *J. VLSI Signal Process. Syst.*, vol. 29, no. 3, pp. 197–207, Nov. 2001.

[5] K. Lahiri, A. Raghunathan, and S. Dey, "Design space exploration for optimizing on-chip communication architectures," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 23, no. 6, pp. 952–961, Jun. 2004.

[6] J. Gong, D. D. Gajski, and S. Bakashi, "Model refinement for hardware-software codesign," in *Proc. Eur. Des. Test Conf.*, Mar. 1996, pp. 270–274.

[7] M. Gasteier, M. Munch, and M. Glensner, "Generation of interconnect topologies for communication synthesis," *Proc. Des. Autom. Test Eur.*, pp. 36–43, Feb. 1998.

[8] T. van Meeuwen, A. Vandecappelle, A. van Zelst, and F. Catthoor, "System-level interconnect architecture exploration for custom memory organizations," in *Proc. Int. Sym. Syst. Synthesis*, Oct. 2001, pp. 13–18.

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

[10] S. Kim, C. Im, and S. Ha, "Schedule-aware performance estimation of communication architecture for efficient design space exploration," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 13, no. 5, pp. 539–552, May 2005.

[11] A. Nandi and R. Marculescu, "System-level power/performance analysis for embedded systems design," in *Proc. Des. Autom. Conf.*, Jun. 2001, pp. 599–604.

[12] J. A. Rowson and A. Sangiovanni-Vincentelli, "Interface based design," in *Proc. Des. Autom. Conf.*, Jun. 1997, pp. 178–183.

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

[14] On-chip coreconnect bus architecture, IBM [Online]. Available: http://www.chips.ibm.com/products/coreconnect/index.html

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

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

[17] F. Frescrua, S. Pielmeier, G. Reali, G. Baruffa, and S. C. Cacopardi, "DSP based OFDM demodulator and equalizer for professional DVB-T receivers," *IEEE Trans. Broadcast.*, vol. 45, no. 3, pp. 323–332, Sep. 1999.

[18] F. Poletti, D. Bertozzi, L. Benini, and A. Bogliolo, "Performance analysis of arbitration policies for SoC communication architectures," *J. Des. Autom. Embedded Syst.*, vol. 8, pp. 189–210, Jun./Sep. 2003.

[19] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.

[20] N. Thepayasuwan and A. Doboli, "Layout conscious bus architecture synthesis for deep submicron systems on chip," in *Proc. Des. Autom. Test Eur.*, Feb. 2004, pp. 10108–10115.

[21] S. Srinivasan, L. Li, and N. Vijaykrishnan, "Simultaneous partitioning and frequency assignment for on-chip bus architectures," in *Proc. Des. Autom. Test Eur.*, Mar. 2005, pp. 218–223.

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

[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. Des. Autom. Conf.*, Jun. 2001, pp. 667–672.

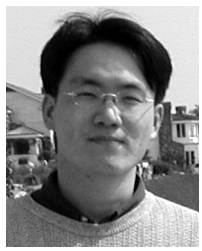
[24] M. Drinic, D. Kirovski, S. Meguerdichian, and M. Potkonjak, "Latency-guided on-chip bus network design," in *Proc. Int. Conf. Comput.-Aided Des.*, Nov. 2000, pp. 420–423.

[25] Wishbone System-on-Chip Interconnection Architecture for Portable IP Cores Silicore and OpenCores [Online]. Available: http://www.opencores.org

[26] W. B. Jone, J. S. Wang, H. Lu, I. P. Hsu, and J. Y. Chen, "Segmented bus design for low-power systems," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 7, no. 1, pp. 25–29, Mar. 1999.

[27] C.-T. Hsieh and M. Pedram, "Architectural power optimization by bus splitting," *Proc. Des. Autom. Test Eur.*, pp. 612–616, Mar. 2000.

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



Sungchan Kim received the B.S. degree in material science and engineering, the M.S. degree in computer engineering, and the Ph.D. degree in electrical engineering and computer science from Seoul National University, Seoul, Korea, in 1998, 2000, and 2005, respectively.

He is presently with Samsung Electronics, Yongin, Gyeonggi, Korea. His research interests include hardware/software codesign, analysis of performance and power consumption, and architecture optimization of SoC for multimedia applications.

Soonhoi Ha (M'94) 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. degree in electrical engineering and computer science from the University of California, Berkeley, in 1992.

He was with Hyundai Electronics Industries Corporation from 1993 to 1994 before he joined the faculty of the School of Electrical Engineering and Computer Science, Seoul National University, where he is currently a Professor. His primary research interests are various aspects of embedded system design including hardware/software codesign and design methodologies.