Efficient Exploration of On-Chip Bus Architectures and Memory Allocation

Sungchan Kim Chaeseok Im Soonhoi Ha

School of Electrical Engineering and Computer Science
Seoul National University
Seoul 151-744, Korea
{ynwie, csim, sha}@iris.snu.ac.kr

ABSTRACT

Separation between computation and communication in system design allows the system designer to explore the communication architecture independently of component selection and mapping. In this paper we present an iterative two-step exploration methodology for bus-based on-chip communication architecture and memory allocation, assuming that memory traces from the processing elements are given from the mapping stage. The proposed method uses a static performance estimation technique to reduce the large design space drastically and quickly, and applies a trace-driven simulation technique to the reduced set of design candidates for accurate performance estimation. Since local memory traffic as well as shared memory traffic are involved in bus contention, memory allocation is considered as an important axis of the design space in our technique. The viability and efficiency of the proposed methodology are validated by two real-life examples, 4-channel digital video recorder (DVR) and an equalizer for OFDM DVB-T receiver.

Categories and Subject Descriptors

C.5.4 [**Computer System Implementation**]: VLSI Systems; C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures (Multiprocessors)-*Interconnection architectures*

General Terms: Performance, Design

Keywords: System-on-a-Chip, Design space exploration, Communication architecture optimization, Memory allocation

1. INTRODUCTION

Insatiable demand of system performance makes it inevitable to integrate more and more processing elements in a single SoC (System-on-a-Chip) to meet the performance requirement. As a new system design paradigm for such high performance SoCs, separation between (1) function and architecture and (2) between communication and computation is advocated by many researchers [1]. 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 of separately specified target architecture.

Separation between computation and communication enables the system designer to explore the communication architecture independently of component (or processing element) selection and mapping. In this paradigm, 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 architectures is explored to determine the optimal one considering the trade-offs between performance, power, cost, and other design objectives. Since the design space of communication architectures is extremely wide, it is critical to develop a very 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][15][16]. The design space we explore, however, is still very wide, formed by multiple axes such as 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. But speed and accuracy are two conflicting goals of performance estimation. A simulation-based method gives accurate estimation results but pays too heavy computational cost to be used for exploring the large design space. So, research based on this method cannot but exploit only a few design axes to reduce the design space significantly. More efficient static performance estimation methods do not model dynamic effects accurately enough to determine the best 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 trace-driven simulation to accurately evaluate the design points in the reduced space and determine a set of pareto-optimal set of bus architectures.

We assume that the 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 traffic as well as shared memory traffic are also involved in bus contention: memory allocation is considered as an important axis of the design space in our technique. On the other hand, most previous works [5][8] have only considered shared memory traffic so that they do not consider memory allocation separately.

The remainder of this paper is organized as follows. In section 2, we explain the overall procedure of the proposed design space exploration technique with an illustrative example. Section 3 reviews some related works. Section 4 and section 5 discuss the details of the proposed technique with a real-life example. We show some experimental results in section 6 and conclude this paper in section 7.

2. OVERVIEW OF PROPOSED EXPLORATION FLOW

For better understanding of the proposed exploration technique, we use an illustrative example of Figure 1. Initially, the system behavior is specified as a block diagram of four functions blocks. The arcs between function blocks show the execution 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.

Figure 1(b) represents a single-bus implementation. Note that one physical memory component is connected to the 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*,

CODES+ISSS'04, September 8-10, 2004, Stockholm, Sweden

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 Figure 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 Figure 2, starts the exploration process with this single-bus architecture that becomes the only element in the "set of architecture candidates" initially.

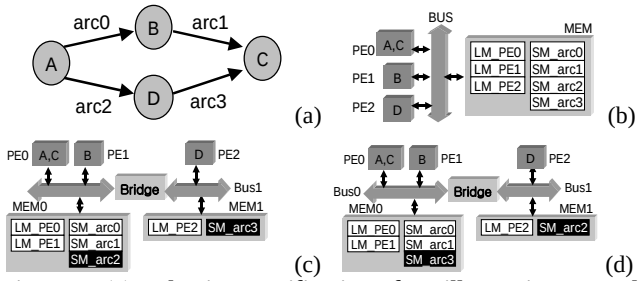


Figure 1. (a) Behavior specification of an illustrative example, (b) its single-bus implementation, (c) and (d) dual-bus implementations, which are different in the mapping of shared memories *SM_arc2* and *SM_arc3.*

Memory traces, including both local and shared memory access traces, from all processing elements is one of the inputs to the proposed exploration procedure. After the mapping of function blocks to processing elements is completed, the memory traces are obtained using cycle-accurate instruction set simulator for each processor core, HDL simulator for ASIC parts, or IP simulators. Memory traces are classified into 3 categories: code memory, data memory and shared memory. Code and data memories are associated with local memory access and shared memory with inter-component communication. In case of processors that are allowed to use cache, the traces associated with code and data memories represent the memory accesses incurred by cache-miss.

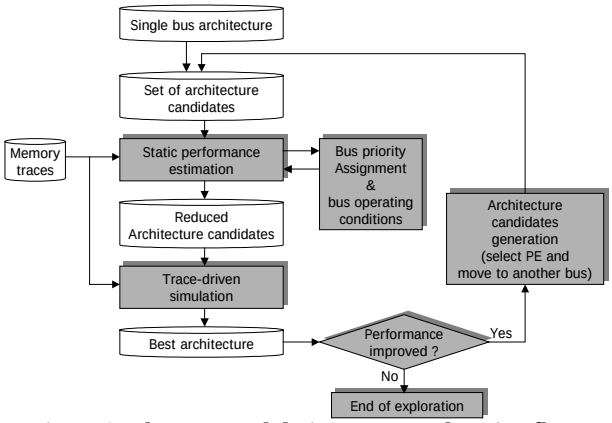


Figure 2. The proposed design space exploration flow.

We traverse the design space in an iterative fashion as shown in Figure 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. 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. We perform static performance estimation to select the design points of top 10% in terms of performance because the proposed static estimation method has less than 10% error around an accurate simulation result [10]. It allows us to prune the design space drastically. The proposed static method is based on the queuing model of the system where the processing elements are the customers and a bus with its associated memory is the 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 detailed explanation is referred to our previous work [10].

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 component and allocating it to a different bus or a new bus. Let us go back to the example of Figure 1. Since there is only one architecture candidate, single-bus architecture, in the first round, 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 4 candidate architectures depending on where to put the shared memory segment 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. Between four candidate architectures Figure 1(c) and Figure 1(d) show two candidate architectures. In 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 more than one architecture, we may explore the wider set of design points with longer execution time.

3. RELATED WORK AND OUR CONTRIBUTION

Some researchers have considered communication architecture selection simultaneously during the synthesis of computation parts of system and 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 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 *et al*. proposed a performance measure technique based on continuous-time Markov processes [11]. However these techniques do not model the dynamic effects such as bus contention and explore only a limited configuration space.

For exploration of communication architectures, simulation-based estimation is also widely adopted in many commercial tools and academic researches at various transaction levels [4][12]. A simulation-based method gives accurate estimation results but pays too heavy computational cost to be used for exploring the large design space. So, research based on this method cannot but exploit only a few design axes to reduce the design space significantly. To overcome this difficulty, a hybrid approach between a static

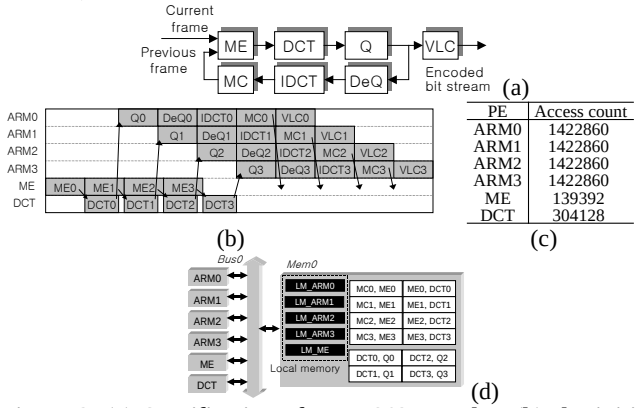
estimation and a simulation approach has been developed by Lahiri *et al*. [3]. They use some static analysis to group the traces and apply a trace-driven simulation with the trace groups. Their approach is similar to ours in that they apply some static analysis to the memory traces to reduce the time complexity of trace-driven simulation.

Since the design space is extremely huge, most previous works focus on a small number of design axes. In Gong *et al*.'s work [6] for example, 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 cost considering only static information such as bit widths and the amount of data transfer and so on [7]. Meeuwen *et al*. presented a technique on cost-efficient interconnect architecture exploration by time-multiplexing the data transfers over a number of shared buses for distributed memory systems under pre-determined memory allocation [8]. Meftali *et al*. find performance-optimal shared memory allocation considering area of communication channel and memory subsystem using 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].

Compared with these related works, there are two main contributions in this paper: First, we explore the larger design space considering multiple design axes such as number of buses, bus topology, component allocation, priority assignment and other bus operating conditions. Since the proposed technique is systematic and extensible, more design axes can easily be added. Second, 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. Since local memory accesses are also involved in bus contention, they need to be considered.

4. DESIGN SPACE EXPLORATION OF 4-CHANNEL DVR SYSTEM

In this section, we explain how the proposed technique explores the design space of a real-life example, 4-channel *DVR* (digital video recorder).



PE	Access count
ARM0	1422860
ARM1	1422860
ARM2	1422860
ARM3	1422860
ME	139392
DCT	304128

Figure 3. (a) Specification of an H.263 encoder, (b) the initial schedule of 4-channel *DVR*, (c) the size of memory traces, and (d) its single-bus implementation.

DVR receives the raw bit streams from external 4 sources and encodes each stream separately using an H.263 encoding algorithm. Figure 3(a) shows the simplified specification of an H.263 encoder. Each H.263 encoder is mapped to a separate ARM720T except all *ME* and *DCT* blocks, which 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. Figure 3(b)-(d) show the scheduling and mapping result of 4-channel *DVR*, the size of memory traces for each processing element, and single-bus implementation of *DVR* respectively. Memory traces are obtained by encoding a P-frame of QCIF-format bit-stream from the same video clip for all channels. This example system has 14 memory segments: 5 local memory segments and 12 shared memory segments. In Figure 3(d), for instance, shared memory segment '*MC0, ME0*' is associated with communication between function blocks *MC0* and *ME0*.

4.1 Communication architecture candidate generation

Performance improvement of communication architecture can be done by scattering communication traffic in conflict on a bus into different buses to reduce conflict and maximize concurrency. For this purpose, we select a processing element and allocate it to a new bus or to another existing bus. Suppose we select *ARM0* for moving in the single-bus architecture of Figure 3. 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 Figure 4. Note that a bus bridge is introduced between two buses for inter-bus communication. For brevity in this paper, we assume that no communication passes through more than 3 buses.

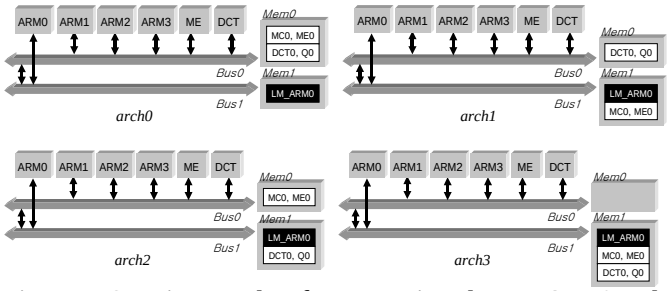


Figure 4. Creating new bus for processing element *ARM0* and allocating its associated shared memory segments.

Table 1. The number of generated architectures from single-bus architecture of 4-channel *DVR* by changing allocation of shared memory segments.

PE being moved	ARM0	ARM1	ARM2	ARM3	ME	DCT
# of shared mem.	2	2	2	2	8	8
# of generated arch.	2^2	2^2	2^2	2^2	2^8	2^8
Total	528					

Table 1 shows how many architecture candidates are generated by moving each processing element from single-bus architecture in Figure 3(d). Overall 528 architecture candidates are generated. This example reveals that how large is the design space to be explored even in the proposed greedy heuristic.

4.2 Bus protocol synthesis of each architecture candidate

Once 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 is worthy to be treated with care since it may cause significant performance variation as observed in [5]. Although an exhaustive search method is guaranteed to be optimal, it is prohibitively expensive. For example, the first architecture, *arch0*, of Figure 4, there are 6 bus masters and 2 bus masters including bus bridges in *bus0* and *bus1* respectively. Therefore, 6! or 1440 priority assignments for *bus0* and 2! assignments for *bus1* are possible so that total 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 the function block fb_i. The criticality $C(fb_i)$ of the function block fb_i is the sum of the schedule length of function blocks on the longest execution path of those that can be executed only after fb_i. The bandwidth and the criticality of a function block are obtained from the memory traces and the function block scheduling. 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

$$R(PE) = \sum_{fb_i \in FB_{PE}} R(fb_i) = \sum_{fb_i \in FB_{PE}} \{BW(fb_i) \cdot C(fb_i)\} \quad (1)$$

where FB_{PE} is the set of function blocks mapped on processing element PE. The higher the rank of a processing element is, a higher priority is assigned to the processing element. After this initial static priority assignment, we swap the priorities of two processing elements on the same bus for further investigation. Although swapping more than two processing elements may yield better results, it incurs longer exploration time. Thus, as a trade-off, we allow swapping of only two processing elements. When the priority assignment of a bus is investigated, assignments of other buses are fixed. For instance, in *arch0* of Figure 4, $\binom{6}{2}$ and $\binom{2}{2}$ assignments are investigated for *bus0* and *bus1* respectively. The proposed assignment heuristic shows remarkable result compared with an exhaustive search method, exploring significantly reduced design space; We validate its efficiency by experimental results in section 6.

Bus clock frequency and bus data-width are dependent on the memory used. Currently we assume that all of the buses in an architecture candidate are synchronized with a single global clock and its frequency is a reciprocal of memory access time for one word. Bus data-width follows the data-width of memory. More sophisticated exploration on these parameters remains as a future work.

5. EXPLORATION ALGORITHM

Figure 5 describes 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.

The best performance values of all architecture candidates are sorted in an ascending order. Then the architecture that has the shortest execution time, *Best_Exe_Time*, is chosen. Since our static estimation method has 10% error bound, 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 line 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 becomes the design space 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 (line 19 to line 25).

The second part of the procedure applies trace-driven simulation to the selected architecture candidates from the first step. We use in-house cycle-accurate trace-driven simulator at this step. Since the estimated performance from 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 buses reaches the number of processing elements, we exit the iteration loop and terminate the procedure.

```
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
```

Figure 5. Proposed exploration flow.

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.

6. EXPERIMENTAL RESULT

In this section, we validate our proposed exploration method by experimental results. All of the experiments were executed at Linux on Xeon 2.8GHz workstation. The first set of experiments is about the heuristic technique for priority assignments: we compared the efficiency of the proposed priority assignment heuristic with the

exhaustive assignment case for the architecture candidates during the exploration of *DVR* system starting from the single-bus architecture in Figure 3(c). To avoid too long run time of exhaustive search, we consider only architectures that have less than four buses. The comparison results are summarized in Table 2. The first row shows the number of searches for a given communication architecture: the proposed heuristic reduces the search space by 181 times on average. The second and the third rows reveal the efficiency of the proposed heuristic. Performance degradation by the proposed heuristic is shown to be negligible. In the 3rd row, how close the heuristic solution approaches to the optimum is represented as the rank of the heuristic solution. If one hundred assignments are possible by the exhaustive method and the proposed heuristic finds the seventh best assignment of them, its rank becomes 7. In short, the smaller the rank is, the closer to the optimum the assignment by heuristic is.

Table 2. Efficiency of the proposed priority assignment heuristic compared with an exhaustive method.

Assignment method	Proposed heuristic			Exhaustive		
	Min.	Max.	Avg.	Min.	Max.	Avg.
# of searches	7	11	9	720	25920	3851
Diff. with exhaustive(%)	0	0.99	0.061	-	-	-
Rank	0	57.08	5.54	-	-	-

In the second set of experiments, the proposed exploration technique is applied to two real-life examples, the *DVR* system and the OFDM equalizer for digital video broadcasting (DVB-T) receivers, and additional 3 example systems. Figure 6 and Figure 7 detail them respectively. In a DVB-T receiver, the equalizer is used for correcting the amplitude distortion of received signals [17]. All function blocks of the equalizer are mapped onto 5 ARM940T processors and are scheduled in pipelined fashion enabling all processors to run concurrently as shown in Figure 6(b). In three additional examples of Figure 7, network of function blocks are generated randomly and function blocks are mapped to arbitrary processing elements. Edges between function blocks assigned to different processing elements denote shared memories. The maximum number of architectures to be evaluated in the second pruning step, *MAX_ARCH* in Figure 5, was fixed to 20. Table 3 represents the results of exploration for each example system. In each set of Table 3, 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 the maximum number of buses that an architecture candidate can have.

The column '*Improvement*' shows the performance improvement, i.e. speedup, of the best architecture among the architecture candidates with the same of number of buses compared with initial single-bus architecture. Performance improvement is hardly obtained near the end of exploration. Most convergence of the best performance appears in the middle stage of exploration. It is noteworthy that the maximum performance of example systems is about twice better than that of the single bus architecture. Since such improvement comes from optimization of only communication architecture and memory allocation, it confirms the viability of the proposed technique. The four rows from the bottom represent the number of total architecture candidates explored, the architecture pruning ratio by static performance estimation of the first exploration step, the average time taken for static performance estimation of an architecture candidate, and the total elapsed time during exploration respectively. In case of *DVR*, pruning ratio is close to 100%. The entire set of architecture candidates includes the ones by the priority assignments as well as the ones 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 terms of an average estimation time in the first step. 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 Figure 8. For all systems, dual-bus system has the widest performance variation meaning that wrong mapping of processing elements or memory allocation could lead significant performance degradation. For example, in *DVR* and *System3*, the worst performance of dual-bus architecture is even inferior to single-bus architecture. However, as more buses are used, its variation becomes smaller since the concurrency of memory accesses is fairly exploited by multiple buses enough to compensate 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.

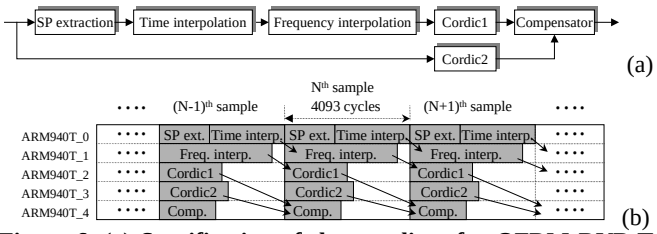


Figure 6. (a) Specification of the equalizer for OFDM DVB-T receiver and (b) its schedule.

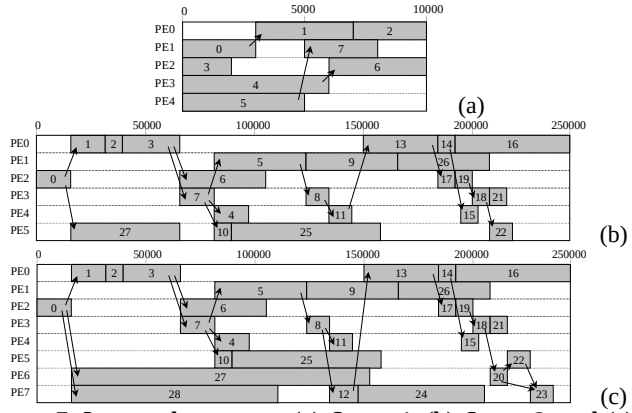


Figure 7. 3 example systems: (a) *System1*, (b) *System2*, and (c) *System3.*

7. CONCLUSION

In this paper, we have presented an iterative two-step exploration technique for optimizing performance of bus-based on-chip communication architecture and memory allocation. At each of iteration, the first step reduces the large design space drastically and quickly by using an efficient static performance estimation method based on queuing model. In the second step, the reduced design space is explored using a trace-driven simulator to choose the best architecture candidate. Experimental results with real-life examples, *DVR* and the equalizer for OFDM DVB-T receiver, and three randomly generated examples validate the efficiency and the viability of the proposed technique to explore the wide design space.

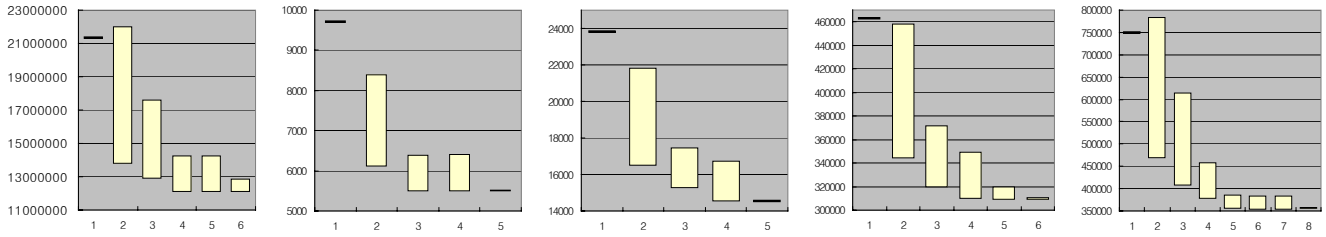
Even though only the performance metric was investigated in this paper, the proposed exploration methodology is extensible to consider other metrics such as power consumption, which is currently under development. Another future work is the extension of the proposed methodology to off-chip system, i.e. board level system.

8. ACKNOWLEDGMENTS

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

Table 3. The experimental results of exploration for 5 example systems including *DVR*.

Application	*DVR*		*Equalizer* for DVB-T		*System1*		*System2*		*System3*	
# of shared mem	14		4		3		14		19	
# of buses	# of arch	Improvement	# of arch	Improvement	# of arch	Improvement	# of arch	Improvement	# of arch	Improvement
1	11	1	7	1	7	1	11	1	22	1
2	5912	1.5511	167	1.5871	119	1.4443	4284	1.3438	15797	1.6022
3	3753	1.6585	107	1.7664	73	1.5592	2537	1.4485	12844	1.8431
4	2119	1.7651	68	1.7661	46	1.6387	1444	1.4921	8424	1.9861
5	1058	1.7686	12	1.7671	6	1.6390	242	1.4974	4903	2.1153
6	260	1.7689					48	1.4963	627	2.1180
7									920	2.1181
8									176	2.1145
Total	13112		360		250		8565		43712	
Pruning ratio(%)	99.6		94.4		93.6		89.1		95.7	
Estimation/arch in 1st step (sec)	0.67		0.06		0.07		0.36		0.82	
Total exe. (sec)	12063		24		17		3239		36234	



(a) *DVR* (b) Equalizer for *DVB-T* (c) *Systrem1* (d) *System2* (e) *System3*

Figure 8. Performance variation of 5 example systems during the exploration according to the number of buses. In each graph, horizontal and vertical axes represent the number of buses used and the estimated execution time in cycles respectively.

9. REFERENCES

[1] K. Keutzer, S. Malik, R. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli, "System-level design: Orthogonalization of concerns and platform-based design", in *IEEE Trans. on Computer-Aided Design*, 19(12), pp.1523-1543, Dec, 2000.

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

[3] K. Lahiri, A. Raghunathan, and S. Dey, "System-level performance analysis for designing system-on-chip communication architecture", in *IEEE Trans. on Computer-Aided Design*, 20(6), pp.768-783, Jun, 2001.

[4] 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 on Signal Processing Systems*, pp.181-190, Oct, 1999.

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

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

[7] M. Gasteier, M. Munch, and M. Glensner, "Generation of interconnect topologies for communication synthesis", in Proc. *Intl. Conf. on. Design Automation and Test in Europe*, 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. *Intl. Symp. on System Synthesis*, pp.13-18, Oct, 2001.

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

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

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

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

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

[14] "IBM On-chip CoreConnect Bus Architecture" http://www.chips.ibm.com/products/coreconnect/index.html

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

[16] "Sonics Integration Architectures, Sonics Inc." http://www.sonicsinc.com

[17] F. Frescrua, S. Pielmeier, G. Reali, G. Baruffa, and S. Cacopardi, "DSP based OFDM demodulator and equalizer for professional DVB-T receivers", in *IEEE Trans. on Broadcasting*, 45(3), pp.323-332, Sep, 1999.