Abstract — In this paper, we are concerned about the performance estimation of bus-based architectures assuming that the task partitioning on the processing components is already determined. Since the communication behavior is usually unpredictable due to dynamic bus requests of processing components, bus contention, and so on, simulation based approach seems inevitable for accurate performance estimation. But it is too time consuming to explore the wide design space. To overcome this serious drawback, we propose a static performance estimation method that is based on the queuing model and makes use of memory traces and task execution schedule information. We propose to use this static estimation approach to prune the design space drastically before applying a simulation-based approach. Comparison with trace-driven simulation results proves the validity of our static estimation technique.

I. INTRODUCTION

In system level design, selection of appropriate communication architecture is a critical design step. In this paper we assume that the system behavior is modeled as a composition of function blocks. Differentiation between functions and communication allows the system designer to explore the communication architecture independently of component selection. Then, communication architecture selection is performed after a decision is made on which processing components are used and which function blocks are mapped on which processing components. And a task is defined as a group of function blocks sequentially scheduled on a processing component.¹

The key tool for exploring the design space of communication architecture is to estimate the performance of communication architectures, which is the main theme of this paper. In particular, we are concerned about the bus-based architectures since the bus-based architecture is most widely adopted due to its simplicity and popularity. From the mapping result of function blocks to the processing elements, we obtain the bus requests from the processing elements and those bus requests are input data to the performance estimation tool.

Since the communication behavior is usually unpredictable due to dynamic trace of memory accesses, bus contention, and so on, simulation based approach seems inevitable for accurate performance estimation. However, since the memory traces of processing elements are enormous, trace driven simulation needs huge data storage for simulating only a few seconds of execution of application. Some commercial tools and research tools use execution-driven simulation where the simulator executes the tasks to produce the bus requests on-line which are passed to the module that accurately models the communication architecture. But the main drawback of a simulation-based approach is that it is too slow to be used for exploring the wide design space of architectures.

Even after a specific bus standard is chosen to be used, the design space can still be huge if a single bus segment cannot accommodate all memory requests within a given time budget. For example, we need to determine how many bus segments are used with what topology and which processing elements and memory banks are mapped to which bus segments. We also have to decide the memory types and memory system configurations. If we include the selection of bus operation frequency and arbitration policy, the design space explodes.

To explore the wide design space, we need a fast and accurate performance estimation method while the simulation-based approach is too slow. We propose a two-step communication architecture exploration scheme to overcome this difficulty. We also propose a static performance estimation method that is based on the queuing model and makes use of memory traces and task execution schedule information. This static estimation approach is used to prune the design space before applying a simulation-based estimation. As the static estimation gets more accurate, the design space to be explored by simulation would be narrower. The key contribution of this paper is to propose an accurate static estimation method taking into account of all dynamic behaviors due to bus contention and dynamic memory traces. We show the effectiveness of our method in terms of estimated time accuracy of entire execution compared with the results of trace-driven simulation.

The remainder of this paper is organized as follows. In section II we state our proposed design space exploration framework and outline the performance estimation techniques. Section III reviews some related works. In section IV and section V the queuing model for a fixed priority based single bus and its extension for modeling round-robin arbitration based bus system are explained respectively. Section VI contains the detailed discussion on the estimation method considering a task schedule and memory traces and on the extension to multiple bus systems. In section VII we give some experimental results and draw conclusions in section VIII.

II. PROPOSED DESIGN SPACE EXPLORATION FRAMEWORK

The proposed design space exploration procedure, depicted in Figure 1(a), assumes that a system behavior is modeled as a composition of function blocks. We do not assume any specific computing model for block composition as long as the system behavior is defined by a non-preemptive execution sequence of function blocks. The only restriction is that the

¹This work was supported by National Research Laboratory Program (number M1-0104-00-0015) and Brain Korea 21 Project. The RIACT at Seoul National University provided research facilities for this study.
A well-known example that meets the restriction. Dataflow specification for digital signal processing (DSP) application is a well-known example that meets the restriction.

![Diagram](image)

**Figure 1.** (a) The proposed design space exploration procedure and (b) the procedure of communication architecture exploration

From the functional specification of system behavior, we first perform platform and component selection and mapping of the function blocks to the processing components. Next, the design space of communication architectures is explored as shown in Figure 1(b). After mapping is completed, we obtain the memory access traces from all processing elements using component simulators: an instruction set simulator for a processor core, HDL simulator for ASIC parts, or IP simulators. In the trace information, we distinguish three different memory access types: code memory, local data memory, and shared memory accesses. We assume that the design space of communication architecture includes memory configurations and memory assignments. Therefore, we include the local memory accesses in the trace information. A memory trace consists of three fields: (processing component, memory access types, time stamp). When we obtain the time stamp, we assume that there is no resource contention, and all memory accesses are finished in a clock cycle.

From the traces, we extract some statistical parameter values to be used in the proposed static estimation method based on the queuing model. Using those parameter values, the static performance estimation of communication architectures is performed to prune the design space. How to prune the design space is beyond the scope of this paper. In the static estimation step, we utilize the task execution schedule on the processing components. The schedule information is used to determine which processing components participate in bus contention at a given instance of time. Finally, trace-driven simulation is performed with a reduced set of candidate communication architecture.

### III. RELATED WORK

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 [6]. Knudsen and Madsen estimated the communication overhead taking into account the data transfer rate variation depending on the protocol, configuration, and different operating frequencies of components [4][5]. However, these techniques do not model the dynamic effects such as bus contention and explore only a limited configuration space.

While simulation-based performance estimation is proposed [2][3] and also widely adopted in the commercial tools at the transaction level [9][10] or at the pin-level [11], there have been proposed only a few static estimation techniques. One is to synthesize the bus architecture with a given and fixed bus request information [7]. But this approach is not applicable in the general case where memory request times are non-deterministic and bus contention exists. A hybrid approach between a static estimation and a simulation approach has been developed by Lahiri et. al. [8]. In their work, communication and computation segments are grouped to make the BSE(Bus and Synchronization Event) graph from the trace data obtained after system cosimulation. They focused on inter-component communication activities that are usually localized in time at the boundary of computation segment. The trace groups are scheduled on the communication media, being shifted by the estimated delays considering the resource contention. They use some static analysis to group the traces and apply a trace-driven simulation with the trace groups. Their approach is similar to ours in that they apply some static analysis to the memory traces to reduce the time complexity of trace-driven simulation. But the overall approach is different from ours. Moreover, they do not consider the local memory accesses. If they consider the local memory accesses, the BSE graph size becomes prohibitively huge.

Our work is inspired by the work in [1] where a simple queuing model of SCSI bus is proposed and the model produces remarkable results comparing with the simulation results.

### IV. QUEUING MODEL OF FIXED PRIORITY BASED SINGLE BUS

In our previous work [14], the estimation technique using a queuing model for fixed priority arbitration based single bus is proposed. The brief explanation of the queuing model of fixed priority based bus system is given in this section. There are \( n \) processing elements, \( PE_0, \ldots, PE_{n-1} \), competing for the use of the bus. \( PE_0 \) has the highest priority. Figure 2 shows the queuing model of single bus architecture. \( \lambda_i \) denotes the rate at which \( PE_i \) issues memory request having no communication architecture overhead. If the execution time is lengthened due to bus contention, the effective arrival rate of request becomes smaller than \( \lambda_i \). We denote the actual memory access rate by \( b_i \) that is actually seen on the bus. The mean service time of a server for the request of \( PE_i \) is denoted by \( 1/\mu_i \). Let \( k_i \) be the expected number of requests from \( PE_i \) waiting for the use of a bus. It is within the range of \([0,1]\) since the processing element
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 can obtain the following equation:

\[
\theta_i = (1-k_i)u_i \lambda_i ,
\]

(1)

where \( u_i = \theta_i/\mu_i \) is the bus utilization of \( PE_i \). Little’s law shows \( w_i = k_i/\theta_i \).

(2)

From this equation, we want to obtain \( \theta_i \), which indicates the delays incurred from contention on the bus. We can extract the parameters \( \lambda_i \) and \( \mu_i \) from the memory traces. There is an unknown parameter \( k_i \). To obtain this parameter, we use the state transition diagram and steady-state probability.

\[ \sum_{j=0}^{i-1} \theta_j = \left( N_i - \sum_{j=0}^{i-1} (k_j + u_j) \right) \gamma_i \]

(3)

The approximate transition rate \( \beta_i \) to state \((n_i,n_{i-1},n_{s})\) is approximated as follows:

\[ \beta_i = \sum_{j=0}^{i-1} \mu_j \cdot u_j / \sum_{j=0}^{i-1} u_j \]

(4)

The similar formulations can be given to the lower priority requests. From the state transition diagram and the transition rates, we can compute the steady-state probability \( p(n_i,n_{i-1},n_{s}) \). After all steady-state probabilities are computed, we obtain the expected number of waiting requests \( k_i \) and consequently the average wait time \( w_i \) by summing up the probabilities of a certain set of states as follows:

\[ \sum_{n_i=n_{i-1}=0}^{\infty} \sum_{n_s=0}^{\infty} p(1,n_i,n_{s}) = u_i + k_i \]

(5)

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

The number of possible transitions from a state grows exponentially as the number of simultaneous events increases. Thus, we make a compromise between the modeling complexity and the modeling accuracy. The compromised model allows up to two simultaneous events. And an event is defined by a single increment or decrement of a coordinate of a state \((n_i,n_{i-1},n_{s})\). For the computation of the transition probabilities, we introduce two new parameters \( d_s \) and \( d_i \) as the probabilities that at least one component issues a new request among high priority processing elements and low priority processing elements respectively:

\[ d_s = 1 - (1-\gamma_s)^{n_{s}-n} \]

(6)

\[ d_i = 1 - (1-\gamma_i)^{n_{i}-n} \]

And let \( S \) the service rate of processing element that is using the bus currently. For instance, the transition probability from state \((n_i,n_{i-1},n_{s})\) to \((n_i,n_{i-1}+1,n_{s})\) becomes \( (1-\lambda_i)d_sSd_{i}(1-S) \). Due to the space limitation of paper, we omit providing all possible state transition rates by considering the simultaneous events. The readers are referred to [14] for more detail of entire queuing model of fixed priority based system.

V. MODELING OF ROUND-ROBIN ARBITRATION BASED SINGLE BUS

Round-robin arbitration is commonly used in on/off-chip bus standards as a mixed arbitration scheme incorporating with other schemes such as fixed-priority or time division multiple access (TDMA) [12][13]. As a preliminary step for modeling mixed arbitration scheme including round-robin arbitration, we propose the queuing model of round-robin arbitration based bus system. Contrary to the fixed-priority arbitration, in round-robin arbitration scheme, the priority of each processing element circulates from the highest priority to the lowest one according to the order which processing element get the bus and currently granted processing element. Thus, intuitively thinking, round-robin arbitration scheme can be thought as begin composed of multiple fixed-priority schemes. By decomposing round-robin based system into multiple fixed-priority based systems and using the probabilities that round-robin system is seen as each fixed-priority system during the entire execution, we get the average wait time for one memory access of a processing element in the round-robin based system.

Figure 3. State transition diagram of round-robin arbitration based bus system
Suppose the round-robin based system that consists of \( n \) processing elements numbered through 0 to \( n-1 \) \((PE_0, \ldots, PE_{n-1})\) and a processing element is granted a bus in ascending order of the processing element number. The state of round-robin based system is defined as an unary \( PL \), which indicates the priority list seen from the bus at certain event sampling point. \( PL \) may have \( n \) values \((pl_0, \ldots, pl_{n-1})\). For example, \( pl_0 \) is the priority list that \( PE_0 \) has the highest priority, which becomes \((0,1,2,\ldots,n-1)\). We also define the steady state probability of state \((PL=pl_i)\) as \( P(PL=pl_i) \) and system, as fixed-priority based system where its priority list is \( pl_i \). Denoting the transition rate from \( system \) to \( system \), as \( \mu_{j,i} \), the state transition diagram is constructed like Figure 3.

Since \( PE_j \) is granted a bus next to \( PE_{j-1} \) by round-robin arbitration scheme, the state transition from \( system \) to \( system \) is caused by the memory access of \( PE_{j-1} \) in \( system \). When \( w_{i,j} \) is the average wait time of \( PE_j \) on \( system \) and the mean service rate of \( PE_j \) is \( \mu_{j,i} \), the transition rate from \( system \) to \( system \), \( \mu_{j,i} \) as follows,

\[
\mu_{j,i} = \frac{1}{w_{i,j} + 1/\mu_{j,i}} 
\]

Using the state transition diagram with the transition rate mentioned above and additional requirement \( \sum_{i=0}^{n-1} P(PL=pl_i) = 1 \), we obtain the steady state probability of each state \((PL=pl_i)\). An average wait time for a memory access of \( PE_j \) on round-robin based system, \( w_{i,j} \), is the sum of the expected average wait time on each fixed-priority based system of \( PE_j \). Since the partial expected average wait time contributed by each fixed-priority based system is proportional to its steady state probability and the actual bus request rate of \( PE_j \) in it, following formula is obtained,

\[
\sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \frac{P(PL=pl_i, \theta_{j,i})}{\theta_{j,i}} = \sum_{i=0}^{n-1} \frac{P(PL=pl_i)}{\theta_{i,j}}
\]

where \( \theta_{i,j} \) is the actual bus request rate of \( PE_j \) on \( system \).

VI. STATIC ESTIMATION USING SCHEDULE INFORMATION

A. Estimation of Single Bus

This section explains how task schedule information is used in our estimation method. A simple statistical modeling of bus requests from a processing component assumes that the bus requests are distributed throughout the whole execution duration of application. This assumption is one of the main sources of inaccuracy of simple static statistical modeling. The schedule information, on the other hand, discloses the active durations of bus requests as shown in Figure 3 (a): for example, it shows that during time 2 and 4, \( PE_2 \) does not generate any bus request.

Therefore, we divide the time slots in such a way that all processing components maintain their bus request patterns during the time slot. And we apply the proposed queuing model analysis in each time slot starting from the beginning of the schedule. Figure 3 displays how the proposed scheme progresses. The shaded region indicates the remaining sections for static estimation. First, we perform the proposed queuing analysis during time 0 to 2 until \( PE_2 \) finishes its execution (Figure 3(a)). Note that the schedule of Figure 3(b) is different from Figure 3(a) since the schedule of the first time slot is updated taking into account the bus contention. In the next time slot, we have to re-compute the statistical parameters for all processing components involved in bus contention.

![Figure 4](image)

Figure 4. (a) Initial schedule of PE0, PE1, and PE2, and corresponding queuing system, (b) new queuing system of PE0 and PE1 after c finished, and (c) another queuing system of PE0 and PE2 in the last time slot

The overall algorithm of the estimation technique for a single bus is described in Figure 5. The procedure \( \text{Estimate_single_bus} \) has 5 inputs. \( B \) represents a data structure of the single bus. \( MT \) and \( \text{ORI_SCHED} \) are the memory traces of all processing elements and the original schedule. \( PL \) is the set of processing components and \( FL \) is the set of tasks to be used for static estimation. The output, \( \text{EVAL_SCHED} \), of this procedure is the updated schedule after considering all bus conflicts and other overheads if any.

**Estimate_single_bus**

**inputs:** \( B, MT, \text{ORI_SCHED}, PL, FL \)

**outputs:** \( \text{SCHEDULE EVAL_SCHED} \)

**begin**

\( \text{EVAL_SCHED} = \text{ORI_SCHED}; \)
\( \text{cur_time} = 0; \)

for each pe \( \in PL \)

\( \text{pe.cur_fb}=\text{NULL}; \)
\( \text{pe.cur_fb_done}=\text{FALSE}; \)

end for

while (FL not empty)

for each pe \( \in PL \)

if (pe.cur_fb_done=TRUE or pe.cur_fb=NULL)

\( \text{pe.cur_fb} = \text{get_current_fb}(\text{EVAL_SCHED}, PL, FL); \)
\( \text{stat_params} = \text{get_stat_params}(\text{ORI_SCHED}, PL, FL, MT); \)
\( \text{if (pe.fb_start_time < cur_time)} \)
\( \text{cur_time} = \text{pe.fb_start_time}; \)

end if

end if

\( \text{cur_time} = \text{estimate_end_time}(B, PL, \text{EVAL_SCHED}, \text{stat_params}); \)
\( \text{update_schedule}(\text{EVAL_SCHED}, \text{cur_time}); \)

end while

**end**

![Figure 5](image)

Figure 5. Schedule-aware performance estimation algorithm for a single bus

The main procedure of queuing analysis is \( \text{estimate_end_time} \). Before the queuing analysis, we first determine which processing components request the bus by calling the \( \text{get_current_fb} \) procedure and compute the statistical parameters by calling the \( \text{get_stat_params} \) procedure. We compute two statistical parameters of each task from the \( MT \) memory trace: they are the memory access rate, \( \theta \), and the mean service time, \( S \). The memory access rate \( \theta_{B,PE} \) is computed by the following formula:
where $M_{FB,PE}$ is a total memory access count during the execution of task FB and $EXE_{FB,PE}$ is the task execution time on PE. When computing the mean service time, we consider the different burst transfer size according to the memory access type. For example, code memory access is usually a burst access of which the size equals to the cache line size whereas data memory access may have various burst lengths for write operation and the cache line size for read. Then the mean service time $S_{FB,PE}$ is computed as

$$S_{FB,PE} = \sum_{i\in\text{all access type}} \left( \sum_{j\in\text{all burst type}} S_{i,j,PE} \frac{M_{i,j,FB,PE}}{M_{FB,PE}} \right),$$

where $S_{i,j,PE}$, $M_{i,j,FB,PE}$ are the service time (including the bus overhead as well as the memory access time) and the number of access counts of burst type $j$ which belongs to memory access type $i$ when task FB runs on processing component PE.

Final procedure update_schedule modifies the original schedule to obtain the updated schedule, EVAL_SCHED, after the current time slot. From the updated schedule, we define the next time slot and go back to the main iteration body until all tasks are considered.

### B. Extension to Multiple Bus System

Communications across buses are achieved via bus bridge in a multiple bus system. A bus bridge plays both roles of a processing component and a memory system as displayed in Figure 6. We assume for simple analysis that no communication passes through more than 3 buses. Figure 6 shows how the bridge is modeled when a processing component on the src bus accesses a memory on the dest bus. The request rate of the component $\lambda_{src}$ is reflected to the dest bus by the request rate $\lambda_{dest}$ of the bridge. To compute $\lambda_{dest}$ we have to know the mean waiting time, $w_{src}$, of the processing component on the src bus. At the same time, the bus bridge looks like a memory from the src bus point of view. The service rate $\mu_{src}$ of the bus bridge should be computed. Let $w_{dest}$ be the mean waiting time of the bridge on the dest bus. Then, $\lambda_{dest}$ and $\mu_{src}$ are computed as follows:

$$\frac{1}{\lambda_{dest}} = \frac{1}{\lambda_{src}} + w_{src} + O_{bridge}, \quad \frac{1}{\mu_{src}} = w_{dest} + O_{bridge} + \frac{1}{\mu_{dest}},$$

where $O_{bridge}$ is overhead time at bus bridge. On the other hand, $w_{src}$ and $w_{dest}$ are obtained from our static estimation technique after $\lambda_{dest}$ and $\mu_{src}$ computation. Therefore the static estimation and bridge modeling are performed iteratively until all parameters become stable.

**Figure 6. The modeling of communication via bus bridge**

### VII. EXPERIMENTS

To investigate the accuracy of the proposed static estimation technique for round-robin arbitration based bus system over a wide variety of working conditions, we first perform the experiments on the example system that has the various number of processing elements that execute one function block concurrently with other processing elements. The memory trace of each processing component is generated with exponentially distributed inter-arrival times between bus requests. We compare the estimation results with those obtained from trace driven simulation. Our trace driven simulator adjusts the time stamps of trace data by accurately modeling the communication architecture that includes buses and memories. More precisely, in these experiments, our bus model consists of 4 phases: bus-arbitration, start-address-drive, sequential-burst-transfer, last-data-drive.

![Figure 7](image.png)

**Figure 7. Estimation accuracy according to the various request rates and the various memory access times**

Table 1 shows the accuracy of estimation on 16 examples systems constructed by the combination of the number of processing elements and the various type of the mean service time. The number of processing elements is one of 2, 4, 6, and 10. The bus request rate($\lambda$) of each processing element is assigned randomly selected value that is between 0.05 and 0.25. When $\lambda$ is 0.1 and memory access time is 1 cycle, the portion of the total memory access time over the entire execution time is about 0.36. This is similar to the case when we run an H.263 encoding algorithm, which we will discuss later. The burst size is also randomly selected during the generation of memory traces. The types of the mean service time are determined by memory access time. Four types of the mean service time are considered, which are the cases where the service time of all processing elements are fixed to 1, 3, and 5 respectively and the case each processing element has arbitrary service time of 1, 3, and 5. We display the range of estimation error with a bar graph, which indicates the minimum and the maximum errors of completion times of processing components. The schedule length of each function block is assumed to 10,000 cycles.

<table>
<thead>
<tr>
<th>Number of processing elements</th>
<th>CPU time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.02</td>
</tr>
<tr>
<td>4</td>
<td>0.11</td>
</tr>
<tr>
<td>6</td>
<td>1.08</td>
</tr>
<tr>
<td>10</td>
<td>18.81</td>
</tr>
</tbody>
</table>

In all cases, the estimation errors of the proposed method are retained within the range from -3% to 4%. This
experiment shows the robust performance of our estimation technique on various architecture configurations.

Table 1 shows the elapsed time for estimating an example system according to the number of processing elements using proposed technique on the Xeon-1.8GHz workstation. We used ILP package to solve the linear equations of fixed priority based bus system constructed by proposed technique. In a certain fixed-priority based system, for each processing component $i$, the number of states $\{(n_i,n_{pi},n_{ui})\}$ is $O(N^2)$ where $N$ is the total number of processing elements. The number of states depends on the priority of the component of interest. Since the time complexity of the LP(linear program) solver is pseudo-polynomial, the overall time complexity for solving a fixed priority based system is also pseudo-polynomial. Consequently the complexity for round-robin based system that consists of fixed-priority based system as much as the number of processing elements is also pseudo-polynomial. Table 1 confirms this fact that the proposed technique has acceptable complexity for design space exploration.

Finally, we validate our proposed technique by applying it to a practical example, 4-channel digital video recorder (DVR). DVR receives the raw bit streams from external 4 sources and encodes each stream separately using H.263 encoding algorithm. Figure 8(a) shows the specification of H.263 encoding algorithm. All function blocks of the H.263 encoder except ME and DCT blocks are mapped to one ARM720T. And all MEs and DCTs are mapped to a motion estimation (ME) hardware block and a discrete cosine transform (DCT) hardware block respectively so that four H.263 encoders share two hardware blocks. Therefore we construct the initial schedule of each function block like Figure 8(b).

With DVR example, we introduce one architecture exploration scenario and verify our proposed estimation technique through it. In this scenario, many architecture candidates are generated by changing the number of buses, the mapping of processing elements to the bus, and associated bridge connections. Other communication architecture parameters, such as priority assignment, bus data-width, and so on, are fixed to the arbitrary values. The results of exploration are summarized in Table 2. The number of buses is changed from one to six. For given number of buses, the number of generated architecture is shown in second column. The number how many bus bridge are included reflects indirectly the complexity of bus topology. In last two columns on right side, the errors of estimation compared with simulation are provided by its range and average of the absolute values. During the exploration over total 200 architectures, our proposed estimation technique keeps its accuracy within the range from -6% to 8.5%. Through this experiment, we verified the accuracy of our proposed technique.

VIII. CONCLUSION

In this paper, we have presented an efficient static estimation technique of round-robin arbitration based bus system. It is based on the queuing model of fixed priority arbitration and makes use of the schedule information and memory traces. Since the estimation time is polynomial to the number of processing components, the number of tasks, and the number of buses, the proposed technique can be used to prune the large design space before trace-driven simulation. Experimental results show our proposed technique performs well providing suitable estimation error, at most 8.5%, compared with trace driven simulation in various communication architecture configurations. Future researches will be focused on the development of design space exploration algorithm and the modeling 2-level TDMA.

REFERENCES