System-level performance analysis of multiprocessor system-on-chips by combining analytical model and execution time variation

Sungchan Kim a,⇑, Soonhoi Ha b

a Division of Computer Science and Engineering, Chonbuk National University, Jeonju, Jeonbuk 561-756, Republic of Korea
b School of Computer Science and Engineering, Seoul National University, Seoul 151-744, Republic of Korea

A R T I C L E   I N F O

Article history:
Available online 18 February 2014

Keywords:
Multiprocessor
System-on-chip
Communication architecture
Performance analysis
Queuing theory

A B S T R A C T

As the impact of the communication architecture on performance grows in a Multiprocessor System-on-Chip (MPSoC) design, the need for performance analysis in the early stage in order to consider various communication architectures is also increasing. While a simulation is commonly performed for performance evaluation of an MPSoC, it often suffers from a lengthy run time as well as poor performance coverage due to limited input stimuli or their ad hoc applications. In this paper, we propose a novel system-level performance analysis method to estimate the performance distribution of an MPSoC. Our approach consists of two techniques: (1) analytical model of on-chip crossbar-based communication architectures and (2) enumeration of task-level execution time variations for a target application. The execution time variation of tasks is efficiently captured by a memory access workload model. Thus, the proposed approach leads to better performance coverage for an MPSoC application in a reasonable computation time than the simulation-based approach. The experimental results validate the accuracy, efficiency, and practical usage of the proposed approach.

© 2014 Published by Elsevier B.V.

1. Introduction

With the ever-increasing complexity of embedded applications, the system complexity is also growing with an increasing number of processing elements in a single chip. Such a chip, which uses multiple processors, is called a Multiprocessor System-on-Chip (MPSoC). Hence, more communication requirements are imposed on on-chip networks, which, in turn, significantly affects the performance of an MPSoC. To cope with such a complexity, designers need to perform a system-level performance analysis in the early stages to explore various design choices before system realization.

Even though simulation-based approaches are popular for estimating on-chip communication performances [26,27], they often suffer from lengthy run times as well as poor performance coverage owing to limited input stimuli or their ad hoc applications. Therefore, recent research on the performance analysis of embedded systems has focused on analytic or semi-formal methods to estimate the worst-case execution time (WCET) [28,32,19,18,33]. In particular, in MPSoC designs, the arbitration policy is a key parameter affecting the performance over various interconnection networks such as a bus, crossbar, or Network-on-Chip (NoC). Fixed priority arbitration is still a popular choice even though it may cause a starvation problem. Related works that have addressed the performance analysis problem on bounded arbitration protocols [30] or unbounded ones [28,32] usually focus on the WCET delay for transferring a network packet or a task-level event stream. Thus, the use of such approaches for a bus transaction-level analysis may result in a severe overestimation such that every bus access undergoes the worst-case arbitration delay. As a result, they are unsuitable for soft real-time system design where the average performance is a primary concern [6,13].

This paper proposes a system-level method to estimate the average performance distribution of an MPSoC with a bus matrix (also known as crossbar switch)-based communication architecture deploying a fixed priority arbitration. A bus matrix provides high throughput while preserving the simplicity of a shared bus abstraction. It is now widely accepted as the industrial de facto standard for on-chip communication in chip multiprocessors [21] as well as in MPSoCs [2,29,31]. A bus matrix has multiple master and slave ports that are connected via multiple internal buses. Any master port can be connected to any slave port in a bus matrix. This is usually referred to as a fully connected matrix [25]. It allows multiple accesses to different slaves in order for them to be in parallel; this results in a higher performance than conventional shared bus architectures. However, there is a scalability issue with regard to the number of master and slave ports in a single bus matrix [35]. One way to resolve the scalability problem is to partially connect
the master and slave ports, thus avoiding resource waste due to unused bus connections.

Packet-switched Network-on-Chip (NoC) architectures are becoming popular as a backbone communication infrastructure that connects processing subsystems and other hardware devices. They combine locally synchronous subsystems with a packet-switched network to build a globally asynchronous system. They may have various topologies such as mesh, ring, and tree, and the regularity of NoC can improve design productivity. The NoC architecture typically allows for a higher clock rate and provides higher communication bandwidth than the bus matrix architecture. However, it has the following disadvantages compared to the bus matrix architecture. First, it incurs the non-negligible overhead of additional buffers and control logsics for converting memory transactions to packets because many IPs are still provided with on-chip bus standards [2,29,31]. Second, packetization/depacketization incurs additional delay. Finally, the latency for delivering packets to a destination over NoC is often more unpredictable than the bus matrix-based architectures because of the complicated transaction protocol of NoC. Hence, the bus matrix architecture is a viable on-chip interconnection scheme for systems with processor of the order of several tens. For large-scale systems with hundreds of processors, typical on-chip interconnection implies a combination of bus matrix for the subsystem and NoC for backbone communication [3,4].

Given a target application and the underlying communication architecture, the proposed technique finds a wider range of performance distribution by corner-case analysis than by a simulation-based approach in significantly less time. The proposed technique consists of two key parts: first, building an analytical model of the target system’s dynamic behavior, and second, systematically exploring, based on the model, the wider performance variations as far as possible within the affordable computation time. The proposed analytical model of a bus matrix architecture is based on the queuing theory and statistics of the memory access behavior of tasks. Then, it is integrated into a unified framework to enumerate task-level execution time variation of a target application, and thereby, to estimate the performance distribution with the underlying communication architecture. Because the execution time variation of tasks constitutes the huge space of execution paths, we propose a scheme to reduce the search space by selecting a representative set of the execution times for a task. In this scheme, the execution time of a task is defined by the memory access count and access request interval.

Experimental results validate the proposed technique. First, our analytical model robustly and accurately predicts the execution time of a target application on various bus matrix architectures. In comparison with the simulation-based approach, the time taken for our analysis is an order of magnitude shorter. Furthermore, the estimated performance on average is 95% accurate. Second, the proposed technique defines a wider performance range than the simulation-based approach along with a faster analysis time by significant orders of magnitude. Experiments over various bus matrix architectures show that the performance ranges obtained by the simulation-based approach lie within the range obtained by the proposed technique. The performance range gap between the two approaches is about 21% on average in terms of the worst/best execution time, which is an acceptable overestimation for practical use. However, it is worth noting that the proposed technique does not guarantee the worst-case performance.

In the next section, we review related work and state our contributions. The overview of the proposed analysis framework is presented in Section 3. Section 4 explains the analytical model of on-chip communication architectures using the queuing theory. Then, in Section 5, the system-level performance analysis technique based on the analytical model is introduced. Experiment results on the accuracy and efficiency of the proposed approach are provided in Section 6. Finally, Section 7 presents the conclusions and addresses future work.

2. Related work

If the communication architecture of an MPSoC is customized to provide a guaranteed latency [9], it is possible to estimate the WCET of an application at the system level. Otherwise, the system performance is usually based on the average performance of the underlying communication architecture as in this study.

Regarding the former case, a considerable number of studies have been conducted on WCET analysis at the system level. In the Modular Performance Analysis (MPA) approach [32], the stream of task-level requests to processing components are modeled as arrival curves. Processing components have their own computational capacities, which are represented as service curves. The stream of requests to a processing component has a bounded delay until the completion of service, which is calculated by the min-plus and max-plus calculi. The various arbitration policies of shared resources are considered at the task level in this approach. The SymTA/S [28] approach proposes a system-level performance analysis method based on the classical schedulability analysis for real-time systems. When the task execution in a component is analyzed by the schedulability analysis, the inter-component interaction is modeled by the abstract event model that is represented as a tuple (p, j, d) indicating the period, jitter, and minimum distance of events. These approaches share a common disadvantage: overestimation caused by high-level abstraction between components.

The approach proposed in [18] uses a discrete event simulation for the schedulability analysis of distributed embedded systems with periodically invoked communicating tasks. The execution scenario of the system is represented by an execution path tree: whenever the scheduling of a new task or the completion of a task having variable execution time occurs, a new node representing a system state is defined and a branch is added to the tree. This exhaustive approach shows better performance coverage than a simulation. Our approach is similar to this approach in that we enumerate the execution paths of an application. However, the proposed technique is different from this work in various ways. First, we consider the contention of the communication architecture for more accurate estimation. Second, we consider the variability of task execution time selectively to reduce the time complexity of the design space exploration.

In order to model the average performance of on-chip networks, several formal approaches have been proposed at various levels of abstractions. Model checking-based approaches have been proposed for AMBA bus verification at the protocol level [19,5]. These approaches, however, require prohibitively huge computation times. Moreover, a nontrivial modeling effort for various communication architecture topologies is required. At the transaction level, on-chip communication architecture models using the queuing theory have been proposed for a hierarchical shared bus [15] and an NoC [10,24,23,7,8,1,14], respectively.

Several studies have proposed analysis techniques for the cross-bar-based interconnections of multi-computer systems [20,22]. They, however, usually ignored the effects of an arbitration policy or focused on bandwidth rather than latency. Therefore, these approaches are unsuitable for the analysis of MPSoC architectures. The work on modeling shared bus or cascaded bus matrices of MPSoCs [15,11] is based on the M/M/1 queuing model or a similar

---

1 Although no quantitative metric of affordable time is given, we regard a technique as affordable if it can be used in practice. For instance, in this study, if an analysis can be done in several hours, we consider it affordable.
analytic model assuming a simple memory architecture. This indicates that they are not generic enough to model various memory access behaviors observed in complicated architectures such as a sophisticated DRAM controller. On the other hand, our approach built on $M/C/1$ queuing model allows us to consider more general memory access patterns, thus enabling more accurate performance predictions as demonstrated by the experiments. In addition, existing research on the automated synthesis of bus matrix-based communication architecture usually assume naive performance models, which consider latency [17] or bandwidth using clock rate and bus width only [25,35,12] thereby ignoring arbitration to resolve resource contention. This study is orthogonal to the work on bus matrix architecture exploration because we focus not on the architecture exploration but on the performance distribution of a given target architecture. For instance, our method may take the candidate architectures, as inputs, from the existing exploration techniques to verify the architectures in more detail.

Although there exist several instances in the literature on the analytic model to predict the performance of NoC, as stated earlier, most of them have focused on latency of a single packet transmission only assuming a constant packet injection rate, or independent packet arrivals [10,24,7,8,1,14]. However, such an assumption may not hold if subsequent packet transmissions take place only after the completion of preceding transmissions, which is a typical assumption for packet transmission to model the memory access behavior of tasks at the system-level design. Hence, the packet injection rate is not constant but subject to change and dependent on packet transmission latency. This implies that the aforementioned NoC models are not directly applicable for our purpose, i.e., system-level performance analysis. Regarding this, the approach in [23] considered the effect of the packet transmission latency on the traffic, or packet injection rate. However, it used a FIFO arbitration scheme, which although unrealistic, is typically assumed in conventional queuing models.

In comparison with previous work, we state the contributions of this study as follows:

1. We introduce a novel analytical model for the performance estimation of the on-chip bus matrix architecture. It provides an accurate and computationally efficient estimation of the average performance.
2. We use the analytical model for the performance distribution analysis of an MPSoC application. Thus, the performance distribution according to the communication workloads of an MPSoC application can be investigated efficiently without the aid of a time-consuming simulation.
3. Because the analytical model supports an arbitrary bus matrix topology, the proposed approach allows us to explore communication architectures.

### 3. Overview of the proposed analysis framework

This section explains the overall flow of the proposed analysis framework. In terms of inputs for the analysis, we assume that a target application is presented as a task graph for the bus matrix-based multiprocessor platform under consideration. Each task may have a variable execution time; task mappings to processors in the platform are known a priori. The objective of the proposed system-level analysis is to estimate the execution time range of the target application for the underlying architecture. We aim at obtaining as wide but tight performance range as possible. We consider the two main causes for performance variation of a target application: variable task execution times and contentsions on communication architecture.

Fig. 1 shows the overall structure of the proposed analysis framework that consists of two stages: statistics profiling and performance distribution analysis. In the statistics profiling stage, we first collect memory access traces of tasks by performing the execution-driven simulation of the target application on the mapped processors. Then, in the next step, i.e., Access pattern profile, the collected traces are translated into statistical information that represents the memory access behaviors of tasks in terms of memory access count and request interval.

Once the memory access behavior is captured, the performance distribution analysis stage is invoked. In this stage, an iterative analysis loop is performed, which consists of two steps. The first step generates an architecture instance, which is a combination of the task execution times sampled within its variations. To cope with the resultant vast design space of architecture instances, we use a simple heuristic to efficiently reduce the search space. In the second step of the analysis loop, i.e., Performance prediction for architecture instance, we deploy an analytic model-based technique instead of a time-consuming simulation to evaluate the performance of the target application.

### 4. Analytical model of on-chip communication architectures

#### 4.1. Baseline analytical model

Fig. 2a shows an application that is specified as a graph $G = (V, E)$, where $V$ is a set of nodes representing tasks or logical memory blocks, and $E$ is a set of edges that represents data dependencies as dotted lines. Logical memories are allocated to physical memories during system realization. While obeying the execution dependencies, a task can start after all preceding tasks have completed. Fig. 2b shows the mapping of tasks to processors (or masters) and logical memory blocks to physical memories (or slaves).

As explained earlier, a bus matrix provides separate access paths to different slaves, thus, allowing concurrent memory accesses. The bus matrix in Fig. 2b has the three slave interfaces where independent arbitrations to the associated slaves occur. We call those interfaces arbitration points (APs). We assume a fixed priority arbitration. In the figure, a lower subscript number indicates a higher priority on the arbitration.

A baseline analytical model represents a system with $M$ masters and a single AP. We are given the following set of parameters to establish a queuing system. In general, these parameters correspond to individual tasks mapped to a master when the master has multiple tasks to run. To simplify the explanation of the model, we assume that a single task is assigned to a master. Note that generality is not lost because of the simplification.

- $C$ denotes row vector $C = [C_0, C_1, \ldots, C_{M-1}]$, where $C_i$ is the memory access count of master $m_i$ to execute a mapped task.
- $\Pi$ denotes row vector $\Pi = [\pi_0, \pi_1, \ldots, \pi_{M-1}]$, where $\pi_i$ is the bus request rate of $m_i$. In addition, $A$ denotes row vector $A = [E[A_0], E[A_1], \ldots, E[A_{M-1}]]$, where $A_i$ is a random variable denoting the bus request interval for $m_i$. The nth moment of random variable $X$ is denoted by $E[X^n]$; the first moment of $X$, $E[X]$, corresponds to its mean. Because a request rate is the reciprocal of an average request interval, $\Pi$ can be rewritten as $\Pi = [\frac{1}{E[X_0]}, \frac{1}{E[X_1]}, \ldots, \frac{1}{E[X_{M-1}]}]$.
- $L$ denotes row vector $L = [E[L_0], E[L_1], \ldots, E[L_{M-1}]]$, where $L_i$ is a random variable for the service time of an access by $m_i$ at a slave with the probability density function $f(l)$, i.e., $f(l) = P(L = l)$. Thus, $E[L_i]$ is the average service time of a bus access by $m_i$. Note that a slave may have different probability

\[\text{Service time is the duration of actual memory access after master } m_i \text{ is granted a bus.}\]
is a random variable for as its diagonal elements. Because we assume be a represents all is a /C1 is accessing until the beginning of a service (an vice time distribution, we model the service times of the server (or intervals. As explained earlier, because the server has a general ser-

Fig. 2. (a) Task model of an application and (b) its mapping to a target architecture.

density functions for different masters. fli(.) can be acquired through various approaches such as simulation, datasheet-based analysis, or timing behavior specification using formal methods.

To model a single AP architecture, we use the M/G/1 queuing system, where M masters are customers, and a slave is a single queuing server. According to Kendall’s notation [16], in the M/G/1 queuing system, ‘G’ denotes the general service time distribution of a server. On the other hand, the previous works [15,10] are based on the M/M/1 queuing system that cannot faithfully model the practical memory subsystem’s behavior, which is usually different from the exponential distribution; it is known as the main source of the modeling inaccuracy.

The transformation of a single bus matrix architecture to its queuing model is depicted in Fig. 3. Processors represent customers and the memory subsystem that includes an AP and a memory represents the server. Customers send requests to the server according to the Poisson distribution with given average request intervals. As explained earlier, because the server has a general service time distribution, we model the service times of the server (or a memory subsystem) separately for each customer (or processor).

We now describe the queuing model for a single AP architecture. All notations and parameters used for the queuing model are summarized in Table 1. First, W denotes a row vector \( W = [E[W_0], E[W_1], ..., E[W_{M-1}]] \), where \( W_i \) is a random variable for the waiting time of master \( m_i \) until the beginning of a service (an actual access to a slave) after a memory access has been requested, i.e., the arbitration delay of the on-chip communication network. Note that the goal of the queuing analysis is to obtain \( W \). We assume that a master can issue a new access request only if its previous request has completed. Although the ideal access rate of master \( m_i \), \( \lambda_i \), is made without any arbitration delay, the actual rate is lower because of the arbitration delay between consecutive accesses. Based on this observation, the effective request rate \( \lambda_i \) of master \( m_i \) is formulated as follows:

\[
\frac{1}{\lambda_i} = E[A_i] + E[W_i] = \frac{1}{\lambda_i} + E[W_i].
\] (1)

To calculate \( W \), we define three more parameters. Let \( N_i \) be a random variable for the number of requests per unit time issued by master \( m_i \); the associated row vector is \( N = [E[N_0], E[N_1], ..., E[N_{M-1}]] \). Because at a time at most one request of a master is observable on a system, \( E[N] \) is the probability that master \( m_i \) requests a memory access at a certain time instance. Then, we have the following formula from Little’s law [16].

\[
E[N_i] = \lambda_i \cdot E[W_i].
\] (2)

Next, we denote the probability that master \( m_i \) is accessing a memory at an instance of time by \( \rho_i \). This parameter is required to represent server utilization in the proposed queuing analysis. Similar to random variable \( N_i \), we define row vector \( B = [\rho_0, \rho_1, ..., \rho_{M-1}] \) for \( \rho_i \). Because Little’s law is also applicable to the server utilization and service time, the following equation holds:

\[
\rho_i = \lambda_i \cdot E[L_i].
\] (3)

Further, as the last parameter in the M/G/1 queuing system, it is required to model the residual service time of the server that is observed when a new access request arrives at the server. For this purpose, we define row vector \( R = [R_0, R_1, ..., R_{M-1}] \), where \( R_i \) is a random variable for the residual service time of an access by master \( m_i \), if the server is processing a request of \( m_i \).

Recalling that the objective of the queuing analysis is to estimate an average arbitration delay of each master, we derive \( E[W_i] \) in terms of the aforementioned parameters. To do this, it is useful to define two \( M \)-by-\( M \) matrices. \( H_i \) and \( O_i \cdot H_i \) represents the masters with higher priorities than \( m_i \) and \( O_i \) represents all masters except \( m_i \) as its diagonal elements. Because we assume
that a master with a lower subscript number takes a higher priority on bus arbitration, \( H_i \) is described as follows:

\[
H_i = (h_{ik})_{k,M+1} = \begin{cases} 1 & \text{if } j = k \land j < i \\ 0 & \text{otherwise} \end{cases}
\]

Similarly, \( O_i = (o_{jk})_{k,M+1} \), where \( o_{jk} = \begin{cases} 1 & \text{if } j = k \land j \neq i \\ 0 & \text{otherwise} \end{cases} \)

Finally, when formulating \( E[W_i] \), we should account for the following cases. First, when master \( m_i \) issues a new request, there may be higher priority masters that have been waiting beforehand. Then, the request of \( m_i \) will be postponed until they are served. The expected waiting time of \( m_i \) in such a case is represented by summing the product of the requesting probability (\( N_i \)) and the associated service time (\( L_i \)) of the higher priority master \( m_j \). The corresponding formulation using the predefined matrices and row vectors is

\[
N(H_i L^T) = N(H_i L^T) + B(O_i R^T) + [E(W_i)IT' (H_i L^T)], \forall i \in \{0, \ldots, M - 1\}.
\]

Rewriting Eq. (6) results in

\[
E[W_i] = \sum_{j=0}^{M-1} E[N_i] E[L_j] + \sum_{j=0}^{M-1} \rho_j E[R_j] + \sum_{j=0}^{M-1} E[W_i] E[L_j], \forall i \in \{0, \ldots, M - 1\}.
\]

In the M/G/1 system, the average residual service time of \( L_i \), \( E[R_i] \), is the function of its first and second moments [16], i.e.,

\[
E[R_i] = \frac{E[L_i^2]}{2E[L_i]^2}.
\]

\( E[L_i] \) and \( E[L_i^2] \) are calculated from \( fll(.) \). Finally, rearranging Eq. (7) with the known terms yields

\[
E[W_i] = \frac{\sum_{j=0}^{M-1} E[W_i] E[L_j] + 0.5 \sum_{j=0}^{M-1} E[L_j]^2}{1 - \sum_{j=0}^{M-1} \rho_j E[L_j]}, \forall i \in \{0, \ldots, M - 1\}.
\]

Using Eq. (9), \( E[W_i] \) can be solved iteratively until all the variables converge to stable values.

### 4.2. Generalization to multiple APs

The baseline analytical model of a single AP system can be extended to a more general bus matrix consisting of \( M \) masters and \( N \) slaves. To this end, we define a few parameters similar to the single AP system analysis. Let us denote a bus request interval of master \( m_i \) as \( \rho_i \) and corresponding random matrix \( A_i \), and let \( L_i \) be a random variable for the service time for an access by \( m_i \) to \( s_j \), and \( W_i \) be a random variable for the waiting time of \( m_i \) until it is served by \( s_j \) after issuing the request. To calculate the effective request rate \( \lambda_{ij} \) for master \( m_i \) to access slave \( s_j \), we should consider the contribution of the request intervals and waiting times caused by accesses to other slaves. Such a consideration is exemplified in 4, where a part of the memory
traces for master \( m_0 \) is provided. Suppose that the system has three slaves, \( s_0, s_1, \) and \( s_2 \). Then, there may be access requests to slave \( s_1 \) or slave \( s_2 \) between two consecutive requests to \( s_0 \), and vice versa. It indicates that the effective request interval of an access to \( s_0 \) becomes longer if accesses to other slaves \( s_1 \) or \( s_2 \) are made in between. Therefore, the contribution of the waiting and service times associated with slaves \( s_1 \) and \( s_2 \) should be taken into account.

To derive the average request interval of a master \( m \) to access slave \( s_j \), we first define \( D_i \) as a random variable indicating a slave that \( m_i \) wants to access with a probability density function \( f_{d_{ij}}(\cdot) \), i.e., \( f_{d_{ij}}(k) = \Pr[D_i = k] \). In addition, we denote by \( V_{ij} \) the portion of accesses to other slaves. For example, \( V_{0j} \) is \( E[L_{0j}] + E[W_{0j}] + E[L_{0j}] + E[W_{0j}] \) in Fig. 4. By generalizing the example above, \( V_{ij} \) is written as

\[
V_{ij} = \sum_{n=0}^{N-1} f_{d_{ij}}(n)[E[L_{ij} + E[W_{ij}]], \forall i \in \{0, \ldots, M_i\}, \forall j \in \{0, \ldots, N - 1\}.
\]

(10)

Then, the formula of \( E_{ij} \) can be extended to the following:

\[
\frac{1}{E_{ij}} = \frac{1}{E_{ij}} + E[L_{ij}] + E[W_{ij}] + E[L_{ij}] + E[W_{ij}]
\]

\[
+ \sum_{n=0}^{N-1} f_{d_{ij}}(n)[E[L_{ij} + E[W_{ij}]], \forall i \in \{0, \ldots, M - 1\}, \forall j \in \{0, \ldots, N - 1\}.
\]

(11)

Once \( E_{ij} \) is calculated, the rest of the analysis for a bus matrix with \( M \) masters and \( N \) slaves is straightforward; similarly, this holds in the case of a single AP system, except that we need to repeat the calculation of Eq. (11) for every slave in the underlying bus matrix. A single iteration of calculating \( E[W_{ij}] \) in Eq. (11) has the time complexity \( O(M \cdot N) \) for an architecture with \( M \) processors and \( N \) APs. Let \( I \) be the number of iterations necessary for the convergence of \( E[W_{ij}] \). Then, the time complexity of the proposed queuing analysis is \( O(I \cdot M \cdot N) \), which is pseudo-polynomial.

4.3. Task-level performance estimation

This section describes a task-level technique to evaluate the execution time of a target application using the analytical model in the previous section. Suppose that an application with six tasks is mapped to three processors on a single memory architecture as depicted in Fig. 5a. Note that the tasks have their own statistical parameters when multiple tasks are mapped to a master. Otherwise, task \( t_1 \) has \( C_i, \lambda_i, L_i, \) and \( W_i \), which should be understood as the parameters associated with the masters in the previous sections. Then, the execution time of task \( t_1, E_i, \) is

\[
E_i = C_i \left( \frac{1}{\lambda_i} + E[L_i] + E[W_i] \right).
\]

(12)

The evaluation begins by solving the queuing system of tasks \( t_0, t_2, \) and \( t_4 \), which are ready to run at time \( t_0 \), as depicted on the leftmost side of Fig. 5b. Then, we calculate the estimated execution time of the three tasks. Note that all the parameters except \( E[W_i] \) are given a priori. On the other hand, \( E[W_i] \) depends on which tasks are running concurrently with task \( t_i \), and it is calculated by the proposed queuing model. Once the execution times of the initially selected tasks are estimated, the earliest estimated execution time among them is chosen, say \( T_i \), for the task \( t_0 \), which is assumed in the figure. Subsequently, at \( T_1 \), a new queuing system is constructed according to the schedule of the remaining tasks. The tasks for the next queuing analysis will be \( t_1, t_2, \) and \( t_4 \), because \( t_0 \) has been completed but the others have not. It is important to consider that the memory access counts for tasks \( t_2 \) and \( t_4 \) at that time are the remainders of the previous execution with \( t_0 \). For instance, if the memory access count \( C_1 \) of task \( t_2 \) at \( t_0 \) was initially 100 and it has carried out 70 accesses during the execution of task \( t_0 \), \( C_2 \) would not be 100, but 30 for the queuing system at \( T_1 \). The second queuing analysis is performed to estimate time \( T_2 \) when task \( t_2 \) completes its execution, and following this, we make a new queuing model at time \( T_2 \) again. Such a procedure is repeated until all the tasks are considered by the queuing analysis. The last completion time of the tasks is the execution time of the application.

The time complexity of the overall task-level performance estimation still remains the pseudo-polynomial because, at most, we repeat the queuing analysis as many times as the number of tasks. Through experiments using a real-life example, we show that the proposed analysis is, in practice, viable for exploring a large design space in a time-efficient manner having the capability of quickly evaluating the performance of bus matrix architectures.

5. System-level performance analysis

5.1. Memory access workload model

In general, the execution time of a task is not constant but subject to change based on various factors such as data dependent memory access pattern, internal control flow of task execution, and preemption by task scheduling. Therefore, the execution time variation of tasks should be taken into account for accurate performance analysis of an MPSoC. In our task model, which assumes the static task schedule and the abstract task representation, we do not make any assumption on internal execution. Note that the internal execution time is reflected as the request interval parameter in the memory access pattern model. According to Eq. (12), the task execution time depends on the variations in memory access count and request rate, while \( E[L_i] \) is fixed once a memory architecture is given. Fig. 6 shows the procedure to profile the memory access count and request rate that are inputs of the proposed queuing model. We first perform execution-driven simulation of a target application with the available input stimuli to generate multiple sets of memory traces for each task. The memory trace of a task is the chronological record of bus accesses accounting for a single run of the task. Note that we ignore any latency incurred at a memory subsystem in this step, i.e., an ideal communication architecture is assumed.

The measurement of a memory access count variation is straightforward; we take the minimum and the maximum access counts among the generated traces for task \( t_i \), which corresponds to the request interval of a master in an architecture with multiple slaves. Fig. 7.
to $C_i$ in Fig. 6. As for the request rate variation, we consider a window that spans an arbitrary number of consecutive memory accesses in a memory trace. In case of hard real-time analysis, we should, for example, examine all possible sizes of the window. The arrival curve of the MPA approach [32], as explained in Section 2, is a good example of this approach. The variation measurement by this approach, however, may be unfeasible for soft real-time applications. Even though it is possible to alternatively consider the average request rates of the memory traces as the request rate variation of the associated task, this approach cannot account for a dynamic behavior inside any single trace of the task. Note that windowing is done on per-trace basis. Then, the maximum and minimum request intervals are computed as the maximum and minimum values of size of window over number of accesses within the window by varying the window size, which is illustrated as the step Extraction of memory access statistics in Fig. 6. The size of windows can be configured by designers. Hence, the memory access behavior of task $t_i$ is captured by the following parameters: $C_i$, a set of access counts for the memory traces of $t_i$, and $VMAX_i$ and $VMIN_i$, the maximum and minimum request intervals profiled for $t_i$. Memory Access Workload, which is based on the parameters mentioned above, captures the memory access behavior of a task.

**Definition 1.** (Memory Access Workload). Memory Access Workload (MAW) of a task represents the memory access behavior of a task by using the request interval and memory access count. We denote by $\Gamma_i$ a set of all possible MAW configurations of task $t_i$.

$$\Gamma_i = \{ (v, c) \mid v \in [VMIN_i, VMAX_i], v \in \mathbb{N}_+, c \in C_i \}$$

The variations in request interval and memory access count in the MAWs, obtained from the window analysis, explicitly represent the dynamics of a task. Combining Eq. (12) and the MAW model enables us to incorporate the effects of arbitration delay and the task execution time variation into the proposed system-level analysis to accurately estimate the performance of a target application.

In Fig. 1, the MAW model of tasks is used to generate architecture instances in the main loop of the proposed system-level performance analysis. An architecture instance is generated by choosing an arbitrary instance of an MAW for each task. However, considering all MAWs for the system-level analysis is impractical because of the enormous design space defined by the MAW combinations. Thus, we propose a heuristic, aiming to provide efficiency and diversity in search space coverage. In the proposed heuristic, we choose four extreme instances among all the MAWs of a task as their representatives. Now, let $\Gamma'_i$ denote the four representative MAWs of the task $t_i$. Then,

$$\Gamma'_i = \{ (v', c') \mid v' \in \{VMIN_i, VMAX_i\}, v' \in \mathbb{N}_+, c' \in \{ \min(C_i), \max(C_i) \} \}$$

For instance, if $\Gamma_i = \{ (v, c) \mid v \in [10, 100], c \in [30, 40, 50, 60] \}$ for task $t_i$, the four representative MAWs $\Gamma'_i$ would be $\{(10,30), (10,60), (100,30), (100,60)\}$. Thus, we consider the four representative MAWs in the proposed system-level performance analysis instead of examining all $91 \times 4 = 364$ MAWs.
In Fig. 7, a simple example is presented, which includes eight tasks, along with their MAWs shown in the second and fourth rows of the table in Fig. 7c. For validation purpose, we compare our approach using the MAW model to an exhaustive approach. However, the total number of combinations may be as high as $4.6 \times 10^{13}$ in this simple example. Hence, we reduce the search space by sampling it by 2; in the case of task $t_0$, for example, the request intervals are sampled as [6,8] from [6,7,8,9], and the access counts as [110,110,110]. MAWs of the other tasks are similarly sampled to reduce the search space. We refer to such a search as pseudo-exhaustive. The two approaches are compared in terms of the WCET of the target application and the total analysis time. Fig. 7b shows that our approach, which corresponds to the column ‘4-points’, results in a WCET less than that of the exhaustive approach by only 5% with a faster analysis time of the order of magnitude four. The shaded rows in Fig. 7c depict the specific bus request rates and access counts corresponding to the WCET in the case of a pseudo-exhaustive search.

This example shows that a set of MAWs with the maximum access counts and largest request intervals for all tasks does not result in the WCET of an application. The worst-case scenario is associated with the minimum request interval or access count for task $t_6$. It is caused by the combined effect of task dependency and arbitration delay. Therefore, we include the minimum values into the representative MAWs to account for the non-monotonic dependency of task dynamics on the WCET if it exists.

It also shows a limitation of the proposed heuristic because the request interval of task $t_4$ lies in between the minimum and maximum values, which is not considered in the 4-point heuristic. We investigate this behavior in more detail by varying the MAWs of $t_4$ with the MAWs of other fixed tasks, as shown in Fig. 8. Fig. 8a shows the case for task $t_0$, which shows a typical behavior that

![Fig. 7. (a) An illustrative example, (b) the resultant worst-case execution times and run times for two search methods, and (c) MAWs of the tasks.](image)

![Fig. 8. (a) and (b) Effects of MAWs on task execution time. (c)-(e) Illustrations on the behavior found in (b).](image)
the execution time increases as either request interval or access count increases. On the other hand, there is a point of inflection in case the request interval of task $t_i$ is 9, as shown in Fig. 8b. The corresponding details are depicted through Fig. 8c–e. For processors $P_1$ and $P_2$, task $t_i$ is overlapped only with $t_i$ when the access count of $t_i$ is less than 106, i.e., the MAW of $t_i$, $\gamma_i$, is (9,102) or (9,104). On the other hand, if $\gamma_i = (9,106)$, $t_i$ is also partially overlapped with $t_i$. Therefore task $t_i$ experiences different sets of tasks for arbitration, which causes an anomalous behavior. In summary, when there is a point of inflection, the proposed 4-point scheme might miss the true WCET of an application.

### 5.2. Execution path enumeration

In this section, we describe a method to explore architecture instances created by MAWs model, corresponding to the architecture instance creation step in Fig. 1. First, an execution path is defined as a sequence of task executions where each task among the tasks, $t_i$, is assigned one representative MAWs $\Gamma_i$. When constructing the graph of execution paths, we create a branch whenever any task completes. Because task $t_i$ initially has the four representative MAWs $\Gamma_i$, the execution path branches out in four ways. Therefore, the number of execution paths grow explosively as the number of tasks increase even with the proposed 4-point MAW selection scheme. To cope with this difficulty, we use a simple DFS (depth-first search) heuristic; we search the space of execution paths by DFS, assuming that a full search may not be possible in the presence of a termination condition for the search.

Despite the memory efficiency benefit of the DFS over BFS (breadth-first search) method, it has a disadvantage in terms of a restricted search diversity. Under tight time constraints, the MAWs of some tasks that are located at higher levels in the hierarchy of a search tree may not be considered at all. In our case, a diversity problem is not critical because there is a general tendency that the execution time increases with an increase in the access counts and request intervals. Thus, we choose an initial set of representative MAWs with the maximum access counts and request intervals for all tasks. The experimental results show that this strategy provides a fast convergence to the acceptable WCET bounds. The best-case execution time (BCET) boundary is explored in a manner similar to the WCET enumeration except that the search begins with the minimum access counts and request intervals of tasks. Subsequently, merging the distributions by separate searches with different initial solutions results in a final distribution comprising both the best- and the worst-case performance bounds.

When the search space of an application is explosive, there should a suitable mechanism to terminate the search in the middle of the analysis. Stipulating the termination condition is done by either limiting the analysis time or setting a convergence rule. In our experiments, we used the latter to examine convergence behavior. More precisely, we terminate the WCET search or the BCET search if the following inequality holds, assuming that a convergence is achieved.

$$\frac{\max(W) - \min(W)}{\max(W)} < \delta.$$  

(13)

where $W$ is a set of WCET (or BCET) values obtained most recently during the WCET (or BCET) search and is a given threshold. The cardinality of $W, |W|$, and $\delta$ are set by the designers.

### 6. Experimental results

The proposed method has been implemented in C++. All experiments were conducted on a workstation having a 3.0-GHz Xeon processor and 4.0-GB main memory with a Linux operating system. Fig. 9 shows a real-life application, 6-channel Digital Video Recorder, or in short 6-ch DVR, that is used in surveillance systems. Each channel takes charge of encoding a raw-bit stream using an H.264 algorithm, which is partitioned into two processors: one for the motion estimation (the node me in the figure) and the other for the remaining tasks of a channel. Channel $c_{th}$ triggers the other channels to start encoding. Three bus matrix architectures are considered with different allocations of logical memory blocks to physical memories, as shown in Fig. 9b. Because the purpose of this study is not optimal memory allocation, we intuitively chose the allocations expecting a distinguishable performance difference among the architectures.

#### 6.1. Accuracy evaluation of analytical model

First, we validate the accuracy of the proposed analytical model. We generated 2000 random traces for each of the tasks, which have varying memory access counts and average request intervals according to the MAWs of tasks in Fig. 9c. The request interval of a task varies following the exponential distribution with a randomly selected mean of the request rates within the associated MAWs. Then, a trace-driven timing simulation was applied to the three bus matrix architectures to evaluate the accuracy of the estimated execution times by the analytical model compared to the simulation results. We also repeated the same experiment using another analytical model similar to ours but based on the $M/M/M$ model. The residual service time (see Eq. (6)) of the $M/M/M$ model, $E[R]_{M/M/M}$, was modeled as $E[R]_{M/M/M} = E[L]$, which is due to the memory less property of the exponential distribution.

For more faithful validation, we used an additional synthetic application of Ref. [15] that is described in Fig. 10a, where 12 processors are busy during the same schedule length, 10,000 cycles to make bus conflicts maximize. The bus request rate $\lambda$ of each processor is chosen randomly within the range from 0.05 to 0.2, as shown in Fig. 10b. For a selected bus request rate, we generated the memory traces of each processor following the Poisson distribution (i.e., the exponentially distributed inter-arrival times between bus requests). When generating the memory traces, we also randomized memory access time between 4 and 20. We used the three architecture templates in Fig. 9b again. For each of the architectures, access frequency to different physical memories were set to follow the Poisson distribution with an average of $\lambda$ for the $12 \times 2$ architecture and $\frac{1}{4}$ for the $12 \times 3$ architecture respectively. The remainder of the comparison procedure is the same as the 6-ch DVR case.

Table 2 shows the accuracy comparison results. For each architecture, the simulated execution times and the estimation error are represented as a range. A negative value of the estimation error corresponds to underestimation, which implies that the analytic method predicts lesser execution times than the simulation. On the other hand, a positive value indicates overestimation. In addition, we provided the mean absolute error for each of the analytic models based on the associated architecture. Given the target architecture $k \in \{12 \times 1, 12 \times 2, 12 \times 3\}$, its mean absolute error $MAE_k$ is calculated as

$$MAE_k = \frac{\sum_{i=1}^{N}[exe_{sim} - exe_{analysis}]}{N}.$$  

(14)

where $exe_{sim}$ and $exe_{analysis}$ are the execution time of an application on architecture $k$ with the $i$th set of generated traces through simulation and our analytic mode, respectively. $N$ is the number of traces sets for each target architecture, which indicates that $N = 2000$ in the experiments.

With the 6-ch DVR application, the average estimation errors throughout the three architectures are at most 4.7% by the pro-
posed analytical model \( (M/G/1) \). On the other hand, the error of the \( M/M/1 \)-based model is at least 31.4%, as appeared in the ‘\( 12 \times 3 \)’ architecture, and the average error is 37.1%. The inaccuracy of the \( M/M/1 \) model comes from the pessimistic assumption of the residual service time, \( E[\tau_i] \). More precisely, because \( E[\tau_i] \) is larger than \( E[\tau_j] \), it always results in an overestimated arbitration delay in the analysis. We observe a similar tendency when using a random application, even though the estimation accuracy of the analytical model is higher than that of the 6-ch DVR. Moreover, as shown in the row ‘Error range’ of the table, the estimated execution times by the proposed analytic model were kept accurate throughout the entire trace sets. The experiments show a robust accuracy of the proposed model as long as statistical traffic modeling is used.

### 6.2. Performance distribution analysis

In the second experiment, we applied the execution path enumeration technique to the 6-ch DVR application to estimate the performance distribution for the three bus matrix architectures. For the purpose of comparison, we also took a random search approach using the trace-driven simulation with randomly generated traces from the MAWs in Fig. 9c similar to the first experiment, which is called RTG (Random Trace Generation), representing the current practice \[26,27\]. In order to get both the best- and the worst-case performance bounds, we performed the proposed search heuristic twice, each of which takes a different initial solution as explained in Section 5.2; one for the worst-case performance bound starting from an initial set of representative MAWs with the maximum access counts and request intervals for all tasks, and another for the best case with the opposite set of MAWs with the minimum access counts and request intervals. Subsequently, the results of the separate searches were merged. The resultant performance coverage according to the target architectures are depicted in Fig. 11. The vertical axis indicates the execution time coverage of the 6-ch DVR example on the three target architectures with the methods applied.
Overall, we observe that the proposed path enumeration discovered wider performance range than the RTG over all the bus architectures. The single memory architecture ‘12/C11’ has the distribution spanning over the range of longer execution times than the other architectures. Although the distributions of the architectures ‘12/C21’ and ‘12/C22’ are similar, the longest execution time of ‘12/C22’ is slightly larger than that of the ‘12/C23’ as reported in Table 3. We observe that the performance distribution obtained by the RTG is immature. It implies that the RTG method searched insufficient design space of architecture instances because every architecture instance needs to be simulated time-costly. This also indicates that the trace generation to discover corner cases is not easy in such a randomized way and the resultant prediction might be optimistic compared to actual performance distribution.

Fig. 12 shows the search behaviors of the WCET bound of our approach and the RTG approach in time. We set the parameters \(|W|\) and \(\phi\) of Eq. (13) to 5 and 0.03, respectively, to construct the termination condition for our search heuristic. In case of the RTG, we set a termination condition with a time budget of 200 h. It should be noted that the time units are different in the two graphs: in seconds and in hours, respectively. The performance coverage by the path enumeration was achieved in an affordable computation time of about 2 h on average, whereas the RTG approach failed to converge within the time budget. As one can see in Fig. 12a, the longest execution time found by the path enumeration converges rapidly within the initial 200 s. More specifically, the sub-graph in Fig. 12a shows that most progress of WCETs emerges in the initial tens of seconds. Although not shown here, we observed that the BCET bound has a similar tendency. As a result, the BCET bound quickly lessened at the early phase of the search beginning from the initial solution. On the other hand, quite a gradual convergence appeared over tens of hours in the RTG method. The graphs showed the ability of the proposed analysis technique to efficiently walk through the enormous search space compared to the simulation-based approach.

To explain more details of the search behaviors depicted in the above figure, Table 3 provides additional information, which includes initial execution time, the worst/best-case bounds for each of the architectures obtained by the methods, and improvement of search results by our approach over the RTG method. The observations are as follows.

First, the path enumeration technique started with an initial architecture instance with longer execution than the RTG approach in all the target architectures. It is worth noting that, in the ‘12/C23’

---

**Table 3**

Numerical comparisons of the approaches.

<table>
<thead>
<tr>
<th>WCET</th>
<th>Bus architecture 12 × 1</th>
<th>12 × 2</th>
<th>12 × 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial execution time (cycles)</td>
<td>Proposed ((I_{\text{proposed}}))</td>
<td>40,360,492</td>
<td>36,777,765</td>
</tr>
<tr>
<td></td>
<td>RTG ((I_{\text{RTG}}))</td>
<td>37,498,119</td>
<td>35,398,893</td>
</tr>
<tr>
<td></td>
<td>Proposed ((W_{\text{proposed}}))</td>
<td>48,621,962</td>
<td>44,834,734</td>
</tr>
<tr>
<td>Search result (cycles)</td>
<td>RTG ((W_{\text{RTG}}))</td>
<td>42,986,404</td>
<td>37,901,645</td>
</tr>
<tr>
<td></td>
<td>Proposed ((W_{\text{proposed}}))</td>
<td>1.21</td>
<td>1.22</td>
</tr>
<tr>
<td>Improvement over initial execution time(^a)</td>
<td>RTG ((W_{\text{RTG}}/I_{\text{RTG}}))</td>
<td>1.07</td>
<td>1.03</td>
</tr>
<tr>
<td>Efficiency over RTG(^a)</td>
<td>(W_{\text{proposed}}/W_{\text{RTG}})</td>
<td>1.13</td>
<td>1.19</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BCET</th>
<th>Bus architecture 12 × 1</th>
<th>12 × 2</th>
<th>12 × 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial execution time (cycles)</td>
<td>Proposed ((I_{\text{proposed}}))</td>
<td>32,691,999</td>
<td>29,054,434</td>
</tr>
<tr>
<td></td>
<td>RTG ((I_{\text{RTG}}))</td>
<td>37,498,119</td>
<td>35,398,893</td>
</tr>
<tr>
<td></td>
<td>Proposed ((W_{\text{proposed}}))</td>
<td>22,883,784</td>
<td>19,479,358</td>
</tr>
<tr>
<td>Search result (cycles)</td>
<td>RTG ((W_{\text{RTG}}))</td>
<td>29,589,651</td>
<td>26,423,946</td>
</tr>
<tr>
<td></td>
<td>Proposed ((W_{\text{proposed}}))</td>
<td>0.7</td>
<td>0.67</td>
</tr>
<tr>
<td>Improvement over initial execution time(^b)</td>
<td>RTG ((W_{\text{RTG}}/I_{\text{RTG}}))</td>
<td>0.84</td>
<td>0.77</td>
</tr>
<tr>
<td>Efficiency over RTG(^b)</td>
<td>(W_{\text{proposed}}/W_{\text{RTG}})</td>
<td>0.77</td>
<td>0.74</td>
</tr>
<tr>
<td>Average search time</td>
<td>Proposed</td>
<td>2.07 h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RTG</td>
<td>Time out with 200-h limitation</td>
<td></td>
</tr>
<tr>
<td># visited architecture instances</td>
<td>Proposed</td>
<td>1.92 × 10(^9)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RTG</td>
<td>4 × 10(^4)</td>
<td></td>
</tr>
</tbody>
</table>

\(^a\) Bigger is better.

\(^b\) Smaller is better.
architecture of the WCET case, the final result of the RTG is yet lower than the initial value of the path enumeration. This proves that our assumption that an initial solution with the MAWs with the maximum access counts and request intervals works as a pertinent entry to help the search find a better (i.e., longer) WCET bound.

Second, the proposed path enumeration found the WCETs longer and the BCETs lesser than their initial solutions by 23% and 32% on average, whereas the RTG approach ended in 3% and 22%, respectively. More importantly, the performance bounds found by the path enumeration is wider than the results of the RTG by at least 13% and 21% on average considering both the WCET and the BCET cases. Even though it is difficult to theoretically assess how viable this performance gap is, the recent survey shows that the typical range of overestimation in the determination of the worst case execution time for real-time applications is between 15% and 50% [34]. This observation implicitly supports the viability of the path enumeration for practical usage.

Finally, the superiority of our approach in terms of wider and thus better performance bounds is mainly due to the availability to visit much more architecture instances than the RTG approach by five orders of magnitude in a time-efficient manner, as indicated in the last two metrics ‘Average search time’ and ‘# visited architecture instances’ of Table 3. Such capability of fast design space search is essential when an exhaustive investigation of corner cases is inevitable for determining the performance bound because the contention at communication architecture incurs the execution time anomaly.

7. Conclusion

In this paper, we presented a novel system-level performance analysis technique for the performance distribution of an MPSoC. The analytical on-chip communication architecture model was proposed, which was then integrated into the execution path enumeration framework. To model the variation in memory access traffics, the selected combinations of request intervals and access counts are considered. The experimental results with the 6-ch DVR example and the synthetic applications showed that our analytical model is very accurate under the various bus matrix architectures and on-chip communication traffic. The proposed execution path enumeration resulted in better performance coverage compared to a time-consuming random simulation approach in less computation time by significant orders of magnitude.

As future work, we plan to extend the analytical model to other arbitration policies and more complicated interconnection networks such as cascaded bus matrices or NoCs. The optimization to reduce the search space of the path enumeration while not sacrificing the diversity remains for future work.

Acknowledgment

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (NRF-2013R1A1A1012715), the Industrial Convergence Source Technology Development Program through the Ministry of Science, ICT and Future Planning, Korea (grant 10044313), and Center for Advanced Image of Chosun University. This research was supported by the MSIP (Ministry of Science, ICT & Future Planning), Korea, under the ITRC (Information Technology Research Center) support program supervised by the NIPA (National IT Industry Promotion Agency) (NIPA-2013-H0301-13-1011).

References


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 currently Assistant professor at Chonbuk National University, Korea. His research interests include various aspects and emerging technologies for parallel computing such as reliable multi-processor system, non-volatile memory-based storage, and Big Data processing.

Soonhoi Ha received the B.S. and M.S. degrees in electronics engineering from Seoul National University, Seoul, Korea, in 1985 and 1987, respectively, and the Ph.D. degree in electrical engineering and computer science from the University of California, Berkeley, in 1992. He is currently Professor at Seoul National University, Korea. He was the program co-chair of CODES + ISSS2006, ASP-DAC2008, and ESTIMedia2005–2006. He has been a member of the technical program committee of several technical conferences including DATE, CODES + ISSS, and ASP-DAC. His primary research interests are various aspects of embedded system design including HW/SW codesign, design methodologies, and embedded software design for MPSoC.