A Fast Trace Aware Statistical Based Prediction Model with Burst Traffic Modeling for Contention Stall in A Priority Based MPSoC Bus

Farhan Shafiq1,a) Tsuyoshi Ishiki1,b) Dongju Li1,c) Hiroaki Kunieda1,d)

Abstract: While Multiprocessor System-On-Chips (MPSoCs) are becoming widely adopted in embedded systems, communication architecture analysis for MPSoCs becomes ever more complex. There is a growing need for faster and accurate performance estimation techniques for on-chip bus architecture. This paper presents a novel fast statistical based bus stall prediction model that enables estimating the effects of bus-contention stall on the cycle-count of an application program on a subject MPSoC architecture. Our technique fills the gap in existing techniques for bus performance estimation, that are either not accurate enough (e.g. static techniques) or too slow to be used in iterative analysis (e.g. cycle by cycle arbitration simulation on every bus access). First we formulate a model named “single blocking model” that models blocking of a single bus request due to a single bus transfer on another bus master at a time. Furthermore we augment this model with a “burst blocking model” that models bus stall incurred due to burst bus transfers. Together these two models give us a very fast way to predict bus stalls on an MPSoC bus. It is assumed that each Processor in the system has a distinct fixed priority, and arbitration is based on priority. The proposed technique makes use of accumulated “workload statistics” to accurately predict the “stall cycle counts” caused due to bus contention. This eliminates the need to simulate arbitration on every bus access, resulting in substantial speed-up. Proposed technique is verified by experiments on applications such as “synthetic traffic generators”, “Newton-Euler dynamic control calculation for the 6-degrees-of-freedom Stanford manipulator benchmark”, “Random sparse matrix solver for electronic circuit simulations benchmark”, “Fast Fourier Transform with 1024 inputs of complex numbers” and “SPEC95 Fpppp which is a chemical program performing multi-electron integral derivatives”. Experimental results show that the proposed method delivers a speed-up factor of 1.33, 1.7, 74 and 6 against the simulation method for the four benchmark applications respectively, while average estimation error is 7% for benchmark application, “Fast Fourier Transform with 1024 inputs of complex numbers” and under 1% for other benchmarks.

Keywords: MPSoC shared bus, bus contention model, performance estimation, statistical model

1. Introduction

System-On-Chips are rapidly moving towards multiprocessor architectures (MPSoCs). With the increasing complexity of the MPSoCs and on-chip communication architecture under short time-to-market, there is a growing need for faster and accurate design techniques. There is also a growing demand for a highly optimized design at the system-level, on MPSoC architecture as well as application software.

The performance and efficiency of MPSoCs depends heavily upon the efficiency of interconnect. Therefore, there is a need for better and faster techniques for analysis and optimization of on-chip interconnect. Currently, a lot of MPSoCs use bus based communication architectures due to their simplicity and popularity such as Core-Connect from IBM [1], AMBA from ARM [2], SiliconBackplane from sonics [3], STBus from STMicroelectronics [4] etc. Whereas MPSoCs with many more Processing Blocks are looking towards employing Networks-on-chip (NoCs) as the communication infrastructure [5], [6]. In this paper we focus on bus-based communication architecture.

Since bus is a shared resource between a number of processing elements, there is bound to be some contention between the processing elements for the use of a bus. Due to the dynamic nature of this contention delay, communication overhead is highly unpredictable. Hence bus contention is a very important aspect of performance estimation.

Static techniques are not able to capture the dynamic nature of arbitration while Simulation based estimation is accurate however it is too slow for iterative analysis. In this paper we try to address this issue by presenting a novel performance estimation technique for on-chip bus based architecture. Our approach uses a statistical based model to accurately predict the dynamic stalls caused due to bus contention for a given application. Statistical models usually assume that bus requests are distributed evenly throughout the whole execution duration of an application however, this assumption is a source of inaccuracy since bus access behavior of actual applications cannot be modeled this way. To overcome

---

1 Department of Communications and Computer Engineering, Tokyo Institute of Technology, Meguro, Tokyo 152–8550, Japan

a) farhanshafiq@vlsi.ee.titech.ac.jp
b) isshiki@vlsi.ee.titech.ac.jp
c) dongju@vlsi.ee.titech.ac.jp
d) kunieda@vlsi.ee.titech.ac.jp
this issue we assume that workload statistics for computation as well as bus traffic on each processing element are known. This assumption complies well with a workload simulation technique for example as presented in Ref. [7]. The estimation technique works such that, for an arbitrary window of “T” cycles, histograms on all bus-workloads and computational-workloads on each PE for a given application are provided to the prediction model which calculates the bus contention stall for each PE. The resulting stall cycle counts are added to the overall cycle counts on each PE. Unlike a simulation approach, which requires arbitration simulation on every bus request, our proposed technique runs once every “T” cycles. Since the value of “T” can be chosen to be big or small in accordance with the total number of cycles, the time required for performance estimation does not increase drastically with increasing number of bus workloads. Moreover, unlike other researches in this area, that solely focus on bus-architecture design space exploration, we aim to use this model to enable application developers to perform “bus-performance aware” application optimization and partitioning and provide a better understanding of the bus performance of an application on a target architecture. Once application optimization is satisfactory and a suitable set of bus architectures and mapping are finalized, simulation techniques can be used hence resulting in a considerably shorter simulation time.

The rest of the paper is organized as follows: Section 2 gives an account of related works, Section 3 gives an overview of the proposed simulation flow, Section 4 and Section 5 explain the developed Single Blocking Model (SBM) and Burst Blocking Model (BBM) respectively, Section 6 describes the calculation flow of the proposed technique, Section 7 explains the un-modeled multi-blocking behavior of contention stall as a potential source of inaccuracy, Section 8 gives a summary of experiments and results. Section 9 outlines the future works and Section 10 concludes the paper.

2. Related Works

Related works cover the works associated with performance estimation techniques for MPSoC bus architectures. Some researchers have focused on static techniques for communication architecture performance estimation. Yen and Wolf propose to estimate the communication delay using the worst-case response analysis of the real-time scheduling [8]. Daveau et al. considered static information like maximum bandwidth of channel and bandwidth of processing elements to estimate performance of interconnect between processing elements [9]. Nandi and Marculescu use continuous-time Markov process technique for performance measurement [10]. Drinic et al. used the profiled statistics of inter-core traffic for core-to-bus assignment [11]. Thepayasuwon and Doboli propose a simulated annealing approach [12]. However these techniques are not able to model bus contention due to its dynamic nature and hence these techniques are not suitable for today’s MPSoC. Simulation based approaches have been more popular for performance estimation. Simulation is performed at various abstraction levels. Loghi et al. used Cycle Accurate models written in SystemC to explore AMBA2 and STBus communication architectures for MPSoCs [13]. Kalla et al. executed traces of component behavior on a Pin Accurate Bus Cycle Accurate simulator and achieved speed up over CA simulation model [14]. Caldri et al. used transaction based bus cycle accurate approach to model AMBA2 AHB, APB using function calls for read/writes. Capturing communication systems using TLMs has been tried due to its standardization [15], [16]. Schriner et al. report a quantitative analysis of speed-accuracy tradeoff of TLM, using the advanced high-performance bus (AHB) as a test case, at different abstraction levels [17]. A simulation-based method gives accurate estimation results but pays too heavy a computational cost with increasing number of bus requests and simulation iterations. To overcome this difficulty, a hybrid approach between a static estimation and a simulation approach has been developed by Lahiri et al. [18]. They used some static analysis to group the traces and apply a trace-driven simulation with the trace groups however their approach converges to trace driven simulation as the memory traces become larger. Kim et al. proposed another hybrid performance estimation approach based on queuing analysis [19]. However, static queuing techniques are inherently insufficient to handle complex bus protocol features. Moreover, due to the use of FSM, number of events/state transitions can explode with increasing PEs. Kawahara et al. propose a simulation method that takes memory access contention into account for evaluation of the execution time of an application program [20]. However, the analysis is not based on actual trace of the program rather UML or statecharts of the program is simulated which results in longer simulation time. Moreover the experiments are performed for only two bus-masters.

3. Overview of Proposed Simulation Flow

Let us explain the simulation flow of our proposed technique for performance estimation. Although the model can be used with any schedule aware simulator where the computational and bus workloads on each PE are known, we use our bus model as an addition to the trace driven workload simulator presented in Ref. [7]. Figure 1 shows an overview of the simulation flow. Program execution trace of an application on a given set of input data is computed, through source-level instrumentation and native code execution and encoded as branch bit-stream. This branch bit-stream is then used to steer workload models inside an MPSoC performance estimation framework. Readers are directed to Ref. [7] for detailed reading. However whenever there is a bus access, the bus stall due to contention cannot be calculated from the workload models and on every bus request, arbitration must be simulated to resolve any bus contention. We propose to eliminate this simulation part and replace it with a statistical prediction model. A comparison of the simulation and prediction method will be shown below. Either way, at the end of the workload simulation, an estimation on the cycle counts is produced. Depending on the performance numbers, application optimization or partitioning improvements can be performed and the above steps can be repeated to estimate bus performance. We aim that the estimation on application cycle-counts using our technique can work well with the Tightly Coupled Thread model [21] which enables programmer to specify system level partitioning directly on reference C code.
3.1 Arbitration Simulation Technique in a Trace Simulator

In the simulation technique, arbitration must be simulated to resolve any bus contention on every bus access. There needs to be a global scheduler inside the trace simulator kernel that dispatches a processor on processor queue to the workload simulator where the processor queue is sorted by processor’s simulation clock, or in the case of a tie, by processor’s priority on the bus access. Multiple bus requests must be arbitrated by cycle-accurate bus simulation that leads to a huge computational load. Working principles of the current bus model is illustrated in Fig. 2. In the trace simulator presented in [7] a Program Trace Graph (PTG) is used for efficient trace retrieval. The PTG consists of nodes and edges such that each PTG-node represents function-start, function-end, branch or call. A PTG-edge connects two PTG-nodes and carries attributes about the program execution information, including cycle count between the two PTG-nodes it connects. On the trace simulator, the workload corresponding to each PTG-edge $e_{ni}$ at processor $PE_i$ ($i = 0, 1, \ldots$) contains at most a single leading “bus workload” $B_i[e_{ni}]$ assigned memory access instruction that generates bus traffic, which is followed by normal “computation workload” $C_i[e_{ni}]$, representing normal instruction executions. Computation workload $C_i[e_{ni}]$ denotes the number of execution cycles on an instruction stream contained in an edge $e_{ni}$. Bus workload $B_i[e_{ni}]$ denotes the number of bus access cycles including accurate bus setup cycles and data transfer cycles, but does not include bus stall cycles which can only be obtained by actual bus simulation. When an edge $e_{ni}$ contains a leading bus workload $B_i[e_{ni}]$, the trace simulator kernel triggers the bus arbitration simulation, and if the bus request is not granted due to occupied bus, bus stall cycle $D_i[e_{ni}]$ obtained from the bus arbitration simulation is added to the processor’s simulation clock.

3.2 Statistical Based Technique in a Trace Simulator

Figure 3 illustrates the bus statistics used in stall cycle prediction that are collected during normal trace simulation. At processor $PE_i$, computation-workload $C_i[e_{ni}]$ and bus workload $B_i[e_{ni}]$ on PTG-edge $e_{ni}$ are simply accumulated on processor’s simulation clock, where bus arbitration simulation at each bus workload is not performed. Statistics of $C_i[e_{ni}]$ and $B_i[e_{ni}]$ are collected at $PE_i$ within a predefined bus prediction interval $T$ cycles. All computation workloads within two consecutive bus workloads are merged as a single interval workload $L_{ij}$ where histogram $h_{Bi}$ for $L_{ij}$ and $h_{Ci}$ on all bus workloads $B_i$ are collected. Figure 4 illustrates the bus-stall prediction flow. For every prediction-window of $T$ cycles, Statistics ($N_i, h_{Li}, h_{Bi}$) are collected at each processor $PE_i$, and used to compute the expected bus stall cycles per request $E[D_i]$. Total bus stall cycle count during the bus prediction interval is predicted as $E[D_i] \cdot N_i$, which is added to the processor’s simulation clock, where $N_i$ is total number of bus workloads within the prediction window $T$.

4. Single Blocking Model (SBM)

4.1 Overview

First we develop a limited model named “single blocking model”. This model assumes that any request on a PE will at most be blocked by only one bus-workload on any other PE. The single blocking model assumes that when a bus request on $PE_i$ is blocked by $b_j$ it will for sure be granted after maximum $B_j$ cycles. This constraint is applicable when there are only two masters competing for the bus and no burst traffic is generated on any of the PEs. However, this assumption would introduce inaccuracy in a system where, (1) for $PE_i$ there are two or more higher priority PEs or (2) there is burst traffic on a higher priority PE. This inaccuracy is because there is a possibility that a request on $PE_i$ is consecutively blocked by $b_1$ or $b_j$ after being blocked by $b_j$ resulting in a longer stall than $B_j$ cycles.

4.2 Key Concepts

4.2.1 Computation Workload

Computation workload $L_i$ denotes the number of execution cycles between two successive bus accesses on a Processing Element $PE_i$ where $i$ indicates the priority of a PE. Let $N_i$ be the total number of computation workloads on $PE_i$. 
4.2.2 Bus Workload
Bus workload $B_i$ denotes the number of bus cycles between two successive computation workloads on $PE_i$. Total number of bus workloads on $PE_i$ are same as $N_i$.

4.2.3 Request Probability and Average Interval Workload
Request Probability is the probability $\lambda_i$ that a bus request $r_i$ occurs at each cycle on $PE_i$. Probability that interval $L_i$ equals $n$ (cycles) is given as:

$$\Pr(L_i = n) = \lambda_i(1 - \lambda_i)^{n-1} \quad (n \geq 1)$$

Here, expectation of interval $L_i$ ($E[L_i]$) is given as:

$$E[L_i] = \sum_{n=1}^{\infty} \Pr(L_i = n) \cdot n = \frac{1}{\lambda_i}$$

Hence request probability is given as:

$$\lambda_i = \frac{1}{E[L_i]} \quad (1)$$

4.2.4 Request Inactivation Probability
On each occurrence of bus workload $B_j$, probability that request $r_i$ does not occur within the duration of $B_j - 1$ cycles on $PE_i$ is called request inactivation probability $y_{ij}$.

4.3 Mathematical Model
From here on we will use following terminologies throughout this document,

$r_i$: a request event by processor $PE_i$; $b_{ij}$: a bus event (with arbitrary length $B_j$) at processor $PE_i$; $blk_{ij}$: a blocking event by bus event $b_{ij}$ on request $r_i$; $t_{ij}(k)$: time difference between the first cycle of $b_{ij}$ and request $r_i$; $E[D_{ij}]$: Expectation of bus stall per request at processor $PE_i$.

First we introduce single blocking bus model such that, bus requests on all $PE_i$ are generated randomly with probability $\lambda_i$. The Binary value $\alpha_{ij}$ describes the fixed priority relationship between processors $PE_i$ and $PE_j$.

$$\alpha_{ij} = \begin{cases} 0 & \text{(priority}(PE_i) < \text{priority}(PE_j)) \\ 1 & \text{(priority}(PE_i) > \text{priority}(PE_j)) \end{cases}$$

4.3.1 Bus Stall Expectation Equation
Figure 5 illustrates that bus event $b_{ij}$ that potentially blocks the request $r_i$ is further categorized into two subclasses namely, (a) Consecutive bus event $bb_{ij}$: bus event $b_{ij}$ at processor $PE_i$ follows immediately after (with no interval) a bus event $b_{ij}$ at processor $PE_j$. Let $S_{ij}$ be the probability of event $bb_{ij}$ i.e. $S_{ij} = \Pr(bb_{ij})$.

(b) Non-consecutive bus event $bb_{ij}$: bus event $b_{ij}$ at processor $PE_j$ follows a bus event $b_{ij}$ at processor $PE_i$ after one or more cycles. Clearly: $S_{ij} = \Pr(bb_{ij}) = 1 - \Pr(bb_{ij}) = 1 - S_{ij}$. Here, among $N_j$ occurrences of event $b_{ij}$ on processor $PE_j$, there are $N_j - S_{ij}$ occurrences of event $bb_{ij}$ and $N_j - (1 - S_{ij})$ occurrences of event $bb_{ij}$.

Furthermore, events $bb_{ij}$ and $bb_{ij}$ are annotated as $bb_{ij}(k)$ and $bb_{ij}(k)$ to denote the specific bus workload length of $b_{ij}$. Also, let $S_{ij}(k) = \Pr(bb_{ij}(k))$ and $S_{ij}(k) = \Pr(bb_{ij}(k))$ be the probabilities of event $bb_{ij}(k)$ and $bb_{ij}(k)$. If we assume that bus events $b_{ij}(k)$ with different lengths $k$ are generated randomly with probability $f_{ij}(k)$, then $S_{ij}(k)$ and $S_{ij}(k)$ are given as

$$S_{ij}(k) = \frac{S_{ij} \cdot f_{ij}(k)}{S_{ij} + f_{ij}(k) - 1}, \quad S_{ij}(k) = \frac{S_{ij}(k) \cdot f_{ij}(k)}{S_{ij}(k) + f_{ij}(k) - 1}$$

Figure 6 illustrates the bus stall delays on event $bb_{ij}(k)$ with specific values of $t_{ij}(k)$. Here, the probability of $t_{ij}(k) = n$ on event $bb_{ij}(k)$ is given as

$$\Pr(t_{ij}(k) = n|bb_{ij}(k)) = \lambda_i \cdot (1 - \lambda_i)^{n-1} \quad (2)$$

Whereas bus stall length is given as:

$$D'_{ij}(k)t_{ij}(k) = n, bb_{ij}(k) = k - n \quad (3)$$

Then the conditional bus stall delay on event $bb_{ij}(k)$ are:

$$E[D'_{ij}(k)|bb_{ij}(k)] = \frac{k \lambda_i - 1 + (1 - \lambda_i)^k}{\lambda_i} \quad (4)$$

Figure 7 illustrates the blocking probabilities and bus stall delays on event $bb_{ij}(k)$ with specific values of $t_{ij}(k)$. The probability of $t_{ij}(k) = n$ and its corresponding bus stall delay are:

$$\Pr(t_{ij}(k) = n|bb_{ij}(k)) = \begin{cases} \lambda_i(1 - \lambda_i)^n & (\alpha_{ij} = 0) \\ \lambda_i(1 - \lambda_i)^{n-1} & (\alpha_{ij} = 1) \end{cases} \quad (5)$$

$$D'_{ij}(k)t_{ij}(k) = n, bb_{ij}(k) = k - n \quad (6)$$
where the effective range of n on the above expressions is $n \leq k - 1$. Then the conditional bus stall delay on event $bb_j(k)$ are:

- $\alpha_{ij} = 0$:
  \[
  E[D'_j(k)|bb_j(k)] = \frac{(k+1)\lambda_i - 1 + (1-\lambda_i)^{k+1}}{\lambda_i}
  \]
- $\alpha_{ij} = 1$:
  \[
  E[D'_j(k)|bb_j(k)] = \frac{kk_i - 1 + (1-\lambda_i)^{k}}{\lambda_i}
  \]

Next, the overall bus stall delay expectation $E[D'_j(k)]$ on all bus events is derived as follows:

\[
E[D'_j(k)] = E[D'_j(k)|bb_j(k)] Pr(bb_j(k)) + E[D'_j(k)|\overline{bb}_j(k)] Pr(\overline{bb}_j(k))
\]

\[
E[D_j(k)] = Q_i E[D'_j(k)]
\]

where $Q_i = \frac{N_i}{N_j}$. Here $D'_j(k)$ is defined against $N_j$ bus events on processor $PE_j$. On the other hand, $D_j(k)$ is defined against $N_j$ bus events on processor $PE_i$. Since $D_j(k)N_j = D'_j(k)N_j$, the factor $Q_i$ is introduced to convert the two values observed on processors $PE_j$ and $PE_i$. Hence using Eqs. (7) and (8) in Ref. (9) and subsequently using Eq. (9) in Ref. (10), Overall $E[D_j(k)]$ for $\alpha_{ij} = 0$ becomes:

\[
E[D_j(k)] = \frac{Q_i f_{bb_j}(k) \lambda_i}{\lambda_i} \left( k \lambda_i - U_{ij}(1 - (1-\lambda_i)^{k}) \right)
\]

Where $U_{ij} = S_{ij} + (1-\lambda_i)(1-S_{ij})$  

And Overall $E[D_j(k)]$ for $\alpha_{ij} = 1$ becomes:

\[
E[D_j(k)] = \frac{Q_i f_{bb_j}(k) \lambda_i}{\lambda_i} \left( k \lambda_i - (1 - (1-\lambda_i)^{k}) \right)
\]

Finally, the overall “bus stall” expectation, $E[D_j]$ on all bus events $b_j$ (with arbitrary length) are:

\[
E[D_j] = \sum_k E[D_j(k)]
\]

Therefore,

\[
E[D_j] = \begin{cases} 
    Q_i \left( E[B_j] - U_{ij}(1 - v_{ij}) \right) \lambda_i & (\alpha_{ij} = 0) \\
    Q_i \left( E[B_j] - (1 - v_{ij}) \right) \lambda_i & (\alpha_{ij} = 1)
\end{cases}
\]

Where, $y_{ij} = \sum_k f_{bb_j}(k)(1 - \lambda_i)^{k-1}$ and $v_{ij} = (1-\lambda_i)y_{ij}$

4.3.2 Consecutive Bus-event Probability

As shown in Fig.8, the event $bb_j(k)$ occurs on the following two cases, (a) When $blk_j(k)$ occurs (bus event $b_j(k)$ blocks a request $r_j$) (b) When request $r_j$ occurs immediately after bus event $b_j(k)$. The probability $S_{ij}$ is given as:

\[
S_{ij} = \begin{cases} 
    Q_i(1 - U_{ij}v_{ij}) & (\alpha_{ij} = 0) \\
    Q_i(1 - v_{ij}) & (\alpha_{ij} = 1)
\end{cases}
\]

4.3.3 Expressions for Calculating Bus Stall

Using values of $S_{ij}$ and $\lambda_i$ from Eqs. (14) and (1) respectively, Eq. (13) becomes,

\[
E[D_j] = \begin{cases} 
    Q_i \left( E[B_j] - U_{ij}(1 - v_{ij}) \right) \lambda_i & (\alpha_{ij} = 0) \\
    Q_i \left( E[B_j] - (1 - v_{ij}) \right) \lambda_i & (\alpha_{ij} = 1)
\end{cases}
\]

Let us now discuss the bus event count ratio $Q_j = \frac{N_j}{N_i}$. Although $N_i$ and $N_j$ are the actual bus event counts observed during the bus predication interval, we need to take into account the fact that these bus event counts resulted while we ignored the bus stall delays, and therefore this will lead to inaccuracy if used directly. In order to include the bus stall delay effects in the bus event count ratio, we define the average bus access interval $G_i$ as:

\[
G_i = E[L_i] + E[B_i] + E[D_i]
\]

where $E[L_i]$ is the interval workload expectation, $E[B_i]$ is the bus workload expectation, and $E[D_i]$ is the bus stall delay expectation:

\[
E[D_i] = \sum_k E[D_j(k)]
\]

Then the bus event count ratio $Q_j$ is calculated as

\[
Q_j = \frac{E[L_j] + E[B_j] + E[D_j]}{E[L_j] + E[B_j] + E[D_j]}
\]

Here, $E[D_i]$ and $E[D_j]$ are the predicted bus stall delays that will be calculated by iterative method.

5. Burst Blocking Model (BBM)

5.1 Overview

The burst blocking model (BBM) extends on the single blocking model such that after a bus-workload on $PE_i$ is finished another bus request can be generated immediately after. We call this event a zero interval event and the zero interval probability $\mu_1$ is introduced. As shown in the Fig.9, on the processor with higher priority $PE_i$, the occurrence of a zero interval workload in effect merges the two successive bus workloads, that is, a bus request $r_j$ on the processor with lower priority, sees a merged bus workload that potentially blocks its request. This means that a bus-request could be blocked by multiple bus-workloads indefinitely. Note that when $\mu_1 = 0$ on all $PE_i$, BBM becomes the same as SBM.

5.2 Mathematical Model

After the termination of a bus workload, a request is immediately generated with probability $\mu_1$, in which case the interval
workload becomes zero cycles. With a probability of $(1 - \mu_i)$, non-zero interval request process (with probability $\lambda_i$) starts.  

Zero interval probability $\mu_i$ is obtained from the collected statistics during the bus prediction interval ($N_i, h_{Li}, h_{Bi}$):  

$$\mu_i = \Pr(L_i = 0) = \frac{h_{Li}(0)}{\sum_m h_{Li}(m)} = \frac{h_{Li}(0)}{N_i}$$  

Then the probability of $L_i = n$ is:  

$$\Pr(L_i = n) = \left\{ \begin{array}{ll}
\mu_i (1 - \mu_i) \lambda_i (1 - \lambda_i)^{n-1} & (n = 0) \\
(1 - \mu_i) & (n \geq 1)
\end{array} \right.$$  

Here, expectation of interval $L_i$ ($E[L_i]$) is given as:  

$$E[L_i] = \sum_{n=0}^{\infty} \Pr(L_i = n) \cdot n = \frac{1 - \mu_i}{\lambda_i}$$  

(1.1)  

5.2.1 Merged Bus-workload  

On the processor with higher priority, the occurrence of a zero interval workload in effect merges the two successive bus workloads, that is, for the processor with lower priority, it sees a merged bus workload that potentially blocks its request. Note that zero interval workloads can in fact occur consecutively as well.  

Let $z_i(m)$ be an event where $m$ consecutive zero interval workloads occur on $PE_i$. Probability of $z_i(m)$ is given as:  

$$\Pr(z_i(m)) = (1 - \mu_i) \mu_i^m$$  

Assuming that bus workloads and interval workloads are totally uncorrelated, let $f_{B_j}(m, k)$ be the probability mass function of the merged bus workloads at $PE_j$ by $m$ consecutive zero interval workloads ($m \geq 1$), which can be derived by the convolution of probability mass functions:  

$$f_{B_j}(m, k) = \sum_n f_{B_j}(m - 1, n) f_{B_j}(k - n)$$  

where  

$$f_{B_j}(0, k) = f_{B_j}(k)$$  

Note that $f_{B_j}(m, k)$ is equivalent to the probability distribution of adding $m + 1$ randomly chosen bus workloads that have the same probability distribution $f_{B_j}(k)$. Among $N_j$ bus workloads, $(1 - \mu_j)N_j$ bus workloads are distributed according to $f_{B_j}(0, k)$, $(1 - \mu_j)\mu_j N_j$ bus workloads are distributed according to $f_{B_j}(1, k)$, $(1 - \mu_j)\mu_j^2 N_j$ bus workloads are distributed according to $f_{B_j}(2, k)$, and so on. Therefore, overall bus workload probability distribution becomes:  

$$f_{B_j}^\ast(k) = \sum_{m=0}^{\infty} (1 - \mu_j) \mu_j^n f_{B_j}(m, k)$$  

Also, the expectation of the merged bus workloads is given as:  

$$E[B_j^\ast] = \sum_k f_{B_j}^\ast(k) \cdot k = \frac{E[B_j]}{1 - \mu_j}$$  

(1.5)  

Here, this extended bus workload distribution applies only to the case $a_{ij} = 0$:  

$$\begin{cases}
S_i(0) = S_i \cdot f_{B_j}(k), & S_i(1) = S_i \cdot f_{B_j}(k) \\
\bar{S}_i(0) = S_i \cdot f_{B_j}(k), & \bar{S}_i(1) = S_i \cdot f_{B_j}(k)
\end{cases} \quad (a_{ij} = 0)$$  

$$\begin{cases}
S_i(0) = S_i \cdot f_{B_j}(k), & S_i(1) = S_i \cdot f_{B_j}(k) \\
\bar{S}_i(0) = S_i \cdot f_{B_j}(k), & \bar{S}_i(1) = S_i \cdot f_{B_j}(k)
\end{cases} \quad (a_{ij} = 1)$$  

5.2.2 Bus Stall Expectation Equation  

Figure 10 illustrates event $bb_{ij}(k)$ and $\bar{bb}_{ij}(k)$ with zero interval requests. On event $bb_{ij}(k)$, zero interval requests can happen only when $a_{ij} = 0$ (otherwise the following bus event $b_{ij}(k)$ does not occur). Also, on event $bb_{ij}(k)$, zero interval requests do not happen for all cases $a_{ij} = 0, 1$. Therefore Eq. (2) is rewritten as  

$$\Pr(a_{ij}(k) = n|bb_{ij}(k)) = \left\{ \begin{array}{ll}
\mu_i & (n = 0) \\
(1 - \mu_i) \lambda_i (1 - \lambda_i)^{n-1} & (n \geq 1)
\end{array} \right.$$  

(2.1)  

Equations (3), (5) and (6) remain unchanged. Similarly for $a_{ij} = 0$, Eq. (4) is rewritten as  

$$E[D_{ij}^\ast(k)|bb_{ij}(k)] = \mu_k + (1 - \mu_k) k \lambda_i - 1 + (1 - \lambda_i)^k \\ = k \lambda_i - (1 - \mu_k) (1 - (1 - \lambda_i)^k) \lambda_i$$  

(4.1)  

And Eq. (7) remains unchanged. Next, for $a_{ij} = 1$, the overall bus stall delay expectation $E[D_{ij}^\ast(k)]$ on all bus events is derived as follows:  

$$E[D_{ij}^\ast(k)] = E[D_{ij}^\ast(k)|bb_{ij}(k)] S_i f_{B_j}(k) + E[D_{ij}^\ast(k)|\bar{bb}_{ij}(k)(1 - S_i)] f_{B_j}(k)$$  

$$= \frac{f_{B_j}^\ast(k)}{\lambda_i} \left( k \lambda_i - U_{ij}^\ast(1 - (1 - \lambda_i)^k) \right)$$  

(11.1)  

where  

$$U_{ij}^\ast = (1 - \mu_i) S_i - (1 - \lambda_i)(1 - S_i)$$  

For $a_{ij} = 1$ the overall bus stall delay expectation $E[D_{ij}^\ast(k)]$ remains the same as Eq. (12).  

Moreover, when $a_{ij} = 0$, since $PE_i$ will only see $(1 - \mu_i)N_j$ effective bus workloads, the bus event count ratio $Q_{ij}$ in this case
is rewritten as:

\[ Q_{ij}^* = Q_i(1 - \mu_j) \quad (\alpha_{ij} = 0) \]

Finally, the overall “bus stall” expectation, \( E[D_{ij}] \) on all bus events \( b_j \) (with arbitrary length) are:

\[
E[D_{ij}] = \sum_k E[D_{ij}(k)] = \begin{cases} 
Q_i^* \left( E[B_i] - U_i \frac{E[L_i]}{(1 - \mu_i)} (1 - v_j) \right) & (\alpha_{ij} = 0) \\
Q_i \left( E[B_i] - \frac{E[L_i]}{(1 - \mu_i)} (1 - v_j) \right) & (\alpha_{ij} = 1) 
\end{cases} \tag{13b}
\]

where, \( y_{ij}^* = \sum_k f_{B_i}(k)(1 - \lambda)^{k-1} \) and

\[ v_j^* = \sum_k f_{B_i}(k)(1 - \lambda)^k = (1 - \lambda)y_j^* \]

Moreover, \( v_j^* \) is in fact a power series of \( \mu_j(1 - \lambda)y_j \)

\[ y_{ij}^* = \sum_k f_{B_i}(k)(1 - \lambda)^{k-1} = \frac{(1 - \mu_j)y_j}{1 - \mu_j(1 - \lambda)y_j} \]

Whereas Eq. (14) is rewritten as,

\[
S_{ij} = \begin{cases} 
Q_{ij}^*(1 - U_j v_j^*) & (\alpha_{ij} = 0) \\
Q_{ij}(1 - v_j) & (\alpha_{ij} = 1) 
\end{cases} \tag{14b}
\]

6. Calculation Flow

Detail of the bus stall delay calculation flow is described below.

1. Coefficient calculations: from the new sets of bus workload statistics \( (N_i, h_{ij}, h_{ji}) \) obtained during the bus prediction period \( T \), interval workload expectation \( E[L_i] \), request probability \( \lambda \), and bus workload expectation \( E[B_i] \) are calculated.

2. Initial bus stall delay values: \( \forall i \in PEset, E[D_i] = 0 \)

3. Iterative refinement: calculations for \( E[D_i] \) are repeated until all bus stall delays \( E[D_i] \) converges. Value of \( E[D_i] \) is updated on each iteration.

At the end the calculated values of \( E[D_i] \) are added to the simulation clock of corresponding PE.

7. Multi-blocking Behavior (Limitation of Proposed Technique)

We define multi-blocking behavior as the event where a single bus request on \( PE_i \) is blocked consecutively by bus-workloads on multiple higher priority PEs. Assuming that a request \( r_i \) on \( PE_i \) is blocked by \( PE_j \), and that there is at least one more bus master \( PE_{ij} \) such that \( \alpha_{ij} = 0 \) and \( \alpha_{it} = 0 \), then there is a possibility that immediately after \( PE_i \) releases the shared bus, a bus-workload on \( PE_j \) wins the bus and \( r_i \) is blocked for another bus-workload on \( PE_j \). Note that this consecutive blocking can happen indefinitely. Figure 11 shows the multi-blocking behavior on a shared bus. This behavior has not been modeled as of yet. This does introduce some inaccuracy in our model, however, for the benchmarks and bus-architecture we applied our technique, the error was under an acceptable level. One of the directions in our future research is to also capture the multi-blocking behavior in our model.

8. Experiments and Results

We demonstrate through experiments the accuracy and speed up of our technique as compared to the simulation based technique. A comparison of the simulation flow for both techniques has been reported in Section 3. First we verify our model by using recorded traffic statistics from a synthetic traffic generator as input to our model. Synthetic traffic generators is a good tool to verify different aspects of our model as we can generate different kinds of traffic as per our requirement. Second, we use Recorded traffic patterns from real benchmark applications as reported by Liu, Xu, et al. [23] to verify the applicability of our technique to real applications.

8.1 Synthetically Generated Traffic

We use a synthetic traffic generator application that generates traffic with certain traffic characteristics as specified in a text file and can be changed easily by hand. The traffic characteristics include (a) Total number of traffic generators (PE) (b) ID of generator (c) Packet Length of generated traffic on each generator (d) Average interval between two successive packets on each generator and (e) probability of a zero interval event (f) total simulation time. Note that the prediction model is oblivious to the contents of the traffic characteristics and treats the traffic generator as any application.

In Fig. 12 we report results for a 2-PE system with a shared bus such that \( \mu = 0 \) for all PE. This system can be modeled by SBM with very small error. Synthetic traffic with different parameters is generated in multiple experiments such that there is no burst traffic. The traffic patterns are varied such that the length of bus-workload ranges from 10%-50% of total execution time.

Typically bus request rates in multimedia applications are around 12% (at the lower end of this range), traffic ranges of 17-20% can be thought of as intensive bus traffic where as 50% would be regarded as extremely intensive bus traffic. The prediction error is reported in Fig. 12, which is under 1%.

Fig. 11 Multi-blocking behavior.

Fig. 12 Avg. prediction error for synthetically generated traffic.
Secondly we repeat the experiments such that burst traffic is generated with a probability $\mu$. The prediction results using Single Blocking Model (SBM) and Burst Blocking Model (BBM) are compared in Fig. 13. As the value of $\mu$ increases, prediction error increases drastically for the SBM model however the BBM is able to capture the burst behavior of the traffic and stays well under 1%.

8.2 Bus Architecture

We use a bus-architecture that contains 4 bus masters connected through a shared bus, in order to maximize bus conflict. Mapping of tasks to Processing Elements is performed such that tasks are assigned with the objective to achieve high utilization of the resources. Readers are directed to Ref. [23] for more details on mapping and scheduling.

8.3 Real Benchmark Applications

Below the benchmark applications being used have been introduced. Table 1 summarizes the workload details of each benchmark.

| Benchmark      | Tasks | Communication Links | Probability $\mu$ |
|----------------|-------|---------------------|------------------|
| ROBOT          | 88    | 131                 | 0.268, 0.202, 0.3, 0.545 |
| SPARSE         | 96    | 67                  | 0.175, 0.113, 0.133, 0.226 |
| FFT           | 534   | 1145                | 0.7, 0.753, 0.76, 0.712 |
| FFT_1024complex | 16384 | 25600               | 0.175, 0.113, 0.133, 0.226 |

The first benchmark used is Newton-Euler dynamic control calculation for 6-degrees-of-freedom Stanford manipulator. It consists of 88 tasks and 131 communication links. Detailed and accurate trace of task execution and communication is recorded during cycle-accurate simulation as Recorded Traffic Pattern (RTP). Our model uses the RTP statistics to predict contention stall as explained before. The observed value of zero-interval probability “$\mu$” is 0.268, 0.202, 0.3 and 0.545 on PE0, PE1, PE2 and PE3 respectively. Figure 14 shows a comparison of simulated and predicted cycle-counts on each PE. As seen in Fig. 14 prediction results are very accurate with an average estimation error of .002% which is almost negligible.

8.3.2 Benchmark2: SPARSE Matrix Solver

The second benchmark used is “Random sparse matrix solver for electronic circuit simulations”. It consists of 96 tasks and 67 communication links. The observed value of zero-interval probability “$\mu$” is 0.175, 0.113, 0.133 and .226 on PE0, PE1, PE2 and PE3 respectively. Figure 15 shows a comparison of simulated and predicted cycle-counts on each PE. Prediction results are very accurate with an average estimation error of .015%. Hence we can say that the quality of estimation is very good.

8.3.3 Benchmark3: FFT-1024-complex

The third benchmark application used is a “Fast Fourier Transform” with 1024 inputs of complex numbers. It consists of 16384 tasks and 25600 communication links. The observed value of zero-interval probability “$\mu$” is 0.175, 0.113, 0.133 and .226 on PE0, PE1, PE2 and PE3 respectively. This benchmark application has a bigger number of tasks and communication links as compared to the previous applications tested. It is expected that the multi-blocking behavior of contention stall would be more observable as explained in “Section 7”. Figure 16 shows a comparison of simulated and predicted cycle-counts on each PE. As we expected, the stall predicted for high priority PE is higher than the simulated stall, similarly stall predicted for lower priority PE.
is lower than the simulated stall. This behavior is as expected since the multi-blocking behavior in reality means that high priority PE wins arbitration more often and the lower priority PE loses arbitration more often than as captured by the BBM model presented in this paper. Modeling the multi-blocking behavior is one of our direction for future work.

8.3.4 Benchmark4: FPPPP

The fourth benchmark application used is “SPEC95 Fpppp” which is a chemical program performing multi-electron integral derivatives. It consists of 334 tasks and 1145 communication links. The observed value of zero-interval probability “µ” is 0.7, 0.733, 0.76 and .712 on PE0, PE1, PE2 and PE3 respectively. As evident from the number of communication links compared to tasks and the zero-interval probabilities on each PE, the traffic is very bursty. Figure 17 shows a comparison of simulated and predicted cycle-counts on each PE. As we can see the prediction result is much more accurate compared to the FFT 1024 complex benchmark even though there is intensive traffic. This fact can be understood by observing the probability µ which is very high in the case of this benchmark and since the BBM model captures the burst traffic behavior, the model is able to predict the cycle-counts more accurately.

8.4 Estimation Error

Figure 18 shows cycle-count estimation error using the proposed estimation technique for 4 benchmark applications. Average estimation error on all PEs is reported. The FFT solver application shows the highest average error of 7%. The reason for high error in the FFT solver is that it is a communication intensive application. Which means all masters are trying to request the bus much more frequently. Hence the probability of multi-blocking behavior is high. Since our proposed model does not capture the multi-blocking behavior yet, we observe relatively higher estimation error.

8.5 Simulation Speedup

Next we compare the simulation times using the proposed estimation technique as opposed to the simulation method. The simulation time for the “arbitration simulation” technique can be calculated as,

\[ T_{\text{sim}} = I \left( \sum_{\text{all PEs}} \frac{\sum N t_{ba}}{I} \right) \]

where, \( t_{ba} \) is the time it takes to simulate one bus access, “N” is the total number of bus workloads on a PE and “I” is the number of iterations a benchmark is run. On the other hand, the simulation time for the proposed “prediction” technique can be calculated as,

\[ T_{\text{prd}} = I \left( \sum_{W} t_m \right) \]

where, \( t_m \) is the time it takes to solve the mathematical equations, W is the number of time-windows and I is the number of iterations a benchmark is run. The speed-up ratio then simply becomes,

\[ \text{Speedup} = \frac{T_{\text{sim}}}{T_{\text{prd}}} = \frac{\left( \sum_{\text{all PEs}} \frac{\sum N t_{ba}}{I} \right)}{\left( \sum_{W} t_m \right)} \]

In most of the experiments \( t_m \) was less than 1 ms and under the precision value of the timer being used so we count it as 1 ms. Hence the time needed for bus contention simulation technique increased with increasing bus requests whereas the time required for proposed prediction technique was dependent on “number of time-windows” as \( t_m \leq 1 \). Table 2 and Fig. 19 report the average speed comparison of our experiments for \( I = 100 \). As seen in the table, \( T_{\text{sim}} \) increases substantially with increasing bus-traffic whereas increase in \( T_{\text{prd}} \) is very low as opposed to \( T_{\text{sim}} \). Consequently the speed-up ratio increases from 1.8 to 74. Note that as
the value of “I” increases, a significant amount of simulation time will be saved using the proposed technique. The method achieves great speedup (74x) in case of FFT. Note that in the expression of speedup $N$ stands out as a pivotal factor i.e. $T_{\text{sim}}$ increases with increasing number of bus-workloads. Comparing the communication links for all benchmark applications shown in Table 1, we can clearly observe that for FFT the number of communication links is much larger. Therefore, the value of $N$ is much larger for FFT as compared to other benchmark applications which results in a larger $T_{\text{sim}}$ and a bigger speed-up. As a general rule, a larger value of $N$ results in a larger $T_{\text{sim}}$. In contrast, length of window can be increased according to $N$, so that the number of windows “$W$” does not increase too much. As a result $T_{\text{pot}}$ is much smaller than $T_{\text{sim}}$, which results in a large speed-up.

8.6 Discussions

The objective for performance estimation techniques is to estimate performance accurately while minimizing the simulation time. Here we describe the significance of our work by comparison with related works comparing simulation speed-up and estimation accuracy.

Authors in Ref. [20] propose a coarse-grained simulation method which takes the effect of memory access contention into account. The input application program is described in Executable UMLs and the simulator calculates the processing time of each UML action. No direct speed-up measures are reported however, they report that the complexity of proposed method increases exponentially with the number of processors hence simulation time would increase exponentially with increasing PEs. Experiments are performed on a 2PE system. Estimation errors are about 3% and 5% except for one program for which error is 33.6%. This error is explained as a consequence of the model being heavily dependent on distribution of the addresses of memory accesses. We note that simulation speed-up becomes worse with increasing PEs and estimation accuracy is highly dependent on traffic patterns of an application. In contrast, for our proposed method simulation time only increases linearly with increasing PEs and due to our trace based approach actual traffic pattern of each program is captured so accuracy is not heavily dependent on traffic pattern. Moreover, for 2PE system we achieve accurate results with under 1% error.

Authors in Ref. [18] proposed a hybrid approach for design space exploration for custom communication architecture combining static estimation and simulation. Communication and computation segments are grouped to make a bus and synchronization event (BSE) graph from trace data. BSE graph size is dependent on the memory traces. Therefore, as the traces become larger, their approach converges to trace-driven simulation. Moreover, since their work focuses on design space exploration, no consideration is given to performance estimation for different applications programs. Since they report results against design space exploration techniques, direct comparisons with our work are not possible. However, we note that due to the fact that BSE graph size is dependent on the memory traces, their technique would take much longer time (close to trace simulation) as compared to our technique.

Authors in Ref. [19] propose a queuing model based approach for design space exploration for bus-based architectures. In experiments they perform design space exploration of a 4-Channel digital video recorder (DVR) and report 6%-8.5% estimation error and a speed-up of several orders on trace-driven simulation. The results of their experiments are acceptable however, the queuing models are not able to capture advanced bus features like split-transaction, out-of-order transaction etc. because it affects the service rates used in the queuing model. We feel that this aspect limits the applicability of their approach to advanced bus-systems. In contrast mathematical equations for analytical models are not based around service rates and with due consideration during derivation, such characteristics can be modeled. For instance by introducing a memory access delay factor in the model such that different values are used for different kinds of bus-transactions.

9. Future Work

Further development of our proposed technique has a couple of directions.

Primarily, the proposed model will be extended to capture Multi-Blocking behavior while maintaining the simulation speed-up. Moreover we want to apply our technique to much more complex bus-architectures with up to 20 bus masters and bridges etc. Secondly, multiple arbitration schemes such as, TDM/Round Robin, Lottery based and Least RecentlyGranted can be modeled.

Furthermore, we aim to develop a cache model that can predict the performance of a bus-architecture for any specific application program.

10. Conclusion

We proposed a new method to fast statistical based bus stall prediction that enables estimating the effects of bus-contention stall on the cycle-count of an application program on a subject MPSoC architecture. Our achievement is developing a statistical based prediction model that uses workload statistics of an application and predicts contention stall with considerable accuracy and delivers a significant speedup. First a limited model called Single Blocking Model is developed which is further augmented with a Burst-Blocking Model. Accuracy of the technique is verified against synthetic traffic generator application. Further experiments are performed on four real benchmark applications for an MPSoC bus-architecture consisting of a shared bus connecting four Processing Elements. The cycle-count estimation for each benchmark is reported along with estimation error and simulation speedup. The average estimation error for low traffic benchmarks as well as bursty traffic benchmark is under 1%, whereas for non-bursty traffic intensive benchmark, the not-modeled multi-blocking behavior introduces inaccuracy up to 7%. On the flip-side, speedup increases drastically with increasing traffic. The speedup ratio ranges from 1.33 for low traffic application up to 74 for traffic intensive application. We conclude that the inaccuracy in estimation for traffic intensive benchmarks is acceptable while the achieved speedup is very significant. Furthermore, we identify modeling of the not-modeled behavior and modeling
other arbitration schemes and bus-architecture features as directions for future work.

References
[1] IBM, IBM On-chip CoreConnect Bus Architecture, available from (www.chips.ibm.com).
[2] ARM, ARM AMBA Specification (rev2.0) (2001), available from (www.arm.com/products/system-ip/amba-specifications.php).
[3] SONICSCIRC, Sonics Integration Architecture, available from (http://sonicsinc.com/).
[4] STMicroelectronics, STBus Communication System: Concepts and Definitions (May 2003).
[5] Jantsch, A. and Tenhunen, H.: Networks on chip, Hingham, MA, USA: Kluwer Academic Publishers (2003).
[6] Benini, L. and DMicheli, G.: Networks on Chips: A New SoC Paradigm, IEEE Computers, pp.70–78 (Jan. 2002).
[7] Isshiki, T., Li, D., Kunieda, H., Isomura, T. and Satou, K.: Trace-Driven Workload Simulation Method for Multi-processor System-On-Chips, DAC, San Francisco, California, USA (2009).
[8] Wolf, W. and Yen, T.: Communication synthesis for distributed embedded systems, ICCAD, San Jose, CA, USA (Nov. 1995).
[9] Daveau, J., Ismail, T.B. and Jerraya, A.A.: Synthesis of system-level communication by an allocation based approach, Int. Symp. System Level Synthesis, Cannes, France (Sep. 1995).
[10] Marculescu, A. and Nandi, R.: System level power/performance analysis for embedded systems design, DAC, Las Vegas, NV, USA (June 2001).
[11] Drinic, M., Kirovski, D., Meguerdichian, S. and Potkonjak, M.: Latency-guided on-chip bus network design, ICCAD, San Jose, California, USA (Nov. 2000).
[12] Thephasayaswan, N. and Doboli, A.: Layout conscious bus architecture synthesis for deep submicron system on chip, DATE, Paris, France (Feb. 2004).
[13] Loghi, M.e.a.: Analyzing On-Chip Communication in a MPSoC Environment, DATE, Paris, France (Feb. 2004).
[14] Kalla, P., Hu, X. and Henkel, J.: A flexible framework for communication evaluation in SoC design, ASP-DAC, Shanghai, China (2005).
[15] Caldri et al.: Transaction-Level Models for AMBA Bus Architecture Using SystemC 2.0, DATE, Munich, Germany (2003).
[16] Ghenassia, F.: Transaction-Level Modeling with SystemC, Springer US (2005).
[17] Schirmer, Gunar and Dömer, R.: Quantitative analysis of the speed/accuracy tradeoff in transaction level modeling, ACM Trans. Embedded Computing Systems (TECS), Vol.8, No.1, p. article 4 (Dec. 2008).
[18] Lahiri, K., Raghunathan, A. and Dey, S.: System-level performance analysis for designing system-on-chip communication architecture, IEEE Trans. Computer-Aided Design. Integrate Circuits Syst., Vol.20, pp.768–783 (June 2001).
[19] Kim, S., Im, C. and Ha, S.: Schedule-aware performance estimation of communication architecture for efficient design space exploration, IEEE Trans. Very Large Scale Integration (VLSI) Systems, Vol.13, No.5, pp.539–552 (May 2005).
[20] Kawahara, R. et al., IBM Research Tokyo, Coarse-grained simulation method for performance evaluation of shared memory system, ASP-DAC, Yokohama (Jan. 2011).
[21] Isshiki, T.: Urfianto, Z., Khan, A., Li, D. and Kunieda, H.: Tightly coupled thread: A new design framework for multiprocessor system-on-chips, DA Symposium 2006 IPSJ Symposium Series (2006).
[22] Shaqf, F., Isshiki, T., Li, D. and Kunieda, H.: A Fast and Highly Accurate Statistical Based Model for Performance Estimation of MPSoC on-Chip Bus, SASMI, Yilan, Taiwan (2015).
[23] Liu, W. et al.: A NoC Traffic Suite Based on Real Applications, IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Chennai, India (July 2011).

Farhan Shafiq received his B.E. degree from National University of Sciences and Technology, Pakistan, and M.E. degree from KTH, Sweden, in 2006 and 2009 respectively. Since 2010 he has been studying for his Ph.D. degree in the Department of Communications and Computer Engineering, Tokyo Institute of Technology. His interests include ESL Design, MPSoC interconnect modeling and simulation techniques, MPSoC programming, HW/SW co-design and VLSI architecture.

Tsuyoshi Isshiki has received his B.E. and M.E. degrees in electrical and electronics from Tokyo Institute of Technology in 1990 and 1992, respectively. He received his Ph.D. degree in computer engineering from University of California at Santa Cruz in 1996. He is currently a Professor at Department of Communications and Computer Engineering in Tokyo Institute of Technology. His research interests include MPSoC programming framework, high-level design methodology for configurable system, bit-serial synthesis, FPGA architecture, image processing, fingerprint authentication algorithms, computer graphics, and speech synthesis. Dr. Isshiki is a member of IEEE CAS, IPSJ and IEICE.

Dongju Li received her Ph.D. degree in Electrical and Electronics from Tokyo Institute of Technology in 1998. She is currently an Associate Professor at Department of Communications and Computer Engineering, Graduate School of Science and Engineering, Tokyo Institute of Technology. Her current research interests include embedded algorithm for fingerprint authentication, fingerprint authentication solution for smart phone. Dr. Li is a member of IEEE CAS and IEICE since 1998.
Hiroaki Kunieda was born in Yokohama in 1951. He received his B.E., M.E. and Dr. Eng. Degrees from Tokyo Institute of Technology in 1973, 1975 and 1978, respectively. He was a Research Associate in 1978 and Associate Professor in 1985, at Tokyo Institute of Technology. He is currently Professor at Department of Communications and Computer Engineering in Tokyo Institute of Technology. He has been engaged in researches on Distributed Circuits, Switched Capacitor Circuits, IC Circuit Simulation, VLSI CAD, VLSI Signal Processing and VLSI Design. His current research focuses on fingerprint authentication algorithms, VLSI Multimedia Processing including Video CODEC, Design for System On Chip, VLSI Signal Processing, VLSI Architecture including Reconfigurable Architecture, and VLSI CAD. Dr Kunieda is a member of IEEE CAS, SP society, IPSJ and IEICE.

(Recommended by Associate Editor: Noriaki Suzuki)