IPSJ Transactions on System LSI Design Methodology Vol.8 26–37 (Feb. 2015)

[DOI: 10.2197/ipsjtsldm.8.26]

Regular Paper

An Efficient Performance Estimation Method for Configurable Multi-layer Bus-based SoC

Salita Sombatsiri[1,a)] Yoshinori Takeuchi[1,b)] Masaharu Imai[1,c)]

Received: June 2, 2014, Revised: August 28, 2014,
Accepted: October 15, 2014, Released: February 12, 2015

Abstract: This paper proposes an efficient performance estimation method for configurable multi-layer bus-based SoC, which evaluates system performance in an early stage of design process. The proposed method uses data flow information obtained from a system-level profiling, an architecture-independent loosely-timed transaction level simulation, and constructs a system-level execution dependency graph. Then, based on each architecture-level model, the architecture-level execution dependency graph is constructed and analyzed to estimate the performance of each architecture. In the analysis, the behavior details of shared buses and multi-layer bus are determined based on the analyzed dynamic bus contention and bus protocols' features. Experiments were conducted by modeling the multi-layer AHB and applying the method to estimate performance of the architectures executing JPEG encoder application. The proposed method estimates the performance of SoC with less than 8% of errors comparing to the results from accurate RTL simulations.

Keywords: electronic system level, configurable multi-layer bus, performance estimation

1. Introduction

The design methodology in System-on-a-Chip (SoC) industries is advancing towards Electronic System Level (ESL) to accelerate time-to-market when designing complex systems. Instead of Register Transfer Level (RTL), ESL allows the design in system-level, where functional requirements are specified with Transaction Level Model (TLM). Typical ESL automates the mapping decisions from functional specification onto an architectural model in an architecture exploration framework to suggest optimal architecture candidates which satisfy design constraints.

In architecture exploration, functional specification described with System-Level Design Language (SLDL) is partitioned into hardware-implemented part and software-implemented part. Then, bus architecture is selected, and design quality, such as performance, area and energy consumption, of each architecture is evaluated in system-level. Various architectural platforms, for example, multi-processor, hierarchical bus and multi-layer bus, are modeled as an architectural model to represent architecture candidates during the exploration.

Performance evaluation is one of the most important parts in the exploration process because SoC architecture greatly affects the design quality. Since various architectural platforms shall be modeled for the exploration and all of the architectures based on the model must be evaluated, a fast performance estimation approach that can be applied to all of those platforms is desired.

This paper proposes an efficient performance estimation method for configurable multi-layer bus-based SoC. The proposed method uses the data flow information obtained from a loosely-timed transaction level simulation in the same way as Ref. [1] to analyze system's behavior. The key features of the proposed method are as follows; (1) Predicting the behavior of shared buses and multi-layer bus during the performance estimation according to the analyzed dynamic bus contention and bus protocols' features. (2) Estimating the performance of various architectures according to the speculated buses' behavior by analyzing an architecture-dependent execution graph. (3) By defining protocol's specific parameters and behavior, the proposed method is applicable to estimating performance of various bus protocols. The method estimates the performance of SoC with less than 8% of errors comparing to the results from accurate RTL simulations.

This paper is organized as follows. In Section 2, the related work is explained briefly. Section 3 describes the MoC, the architectural model and the definition of the proposed performance estimation method. Proposed method for performance estimation is explained thoroughly in Section 4. Then, the AMBA AHB's behavior modeling is depicted as an example and the experiments are conducted in Section 5. Finally, this paper concludes with summary and future work.

2. Related Work

It is very complicated to estimate performance of the multi-layer bus-based platform because it involves a large number of signals, including the occasionally active ones. The most common way to evaluate their performance is hardware-software co-simulation [2], [3]. Although these approaches can evaluate every architectural platform, it takes an unacceptably long time because a complete behavior of all possible architectures are modeled and simulated. Hence, models for architecture-level simulation [4], [5], and fast co-simulation models [6], [7], [8] are pro-

[1] Graduate School of Information Science and Technology, Osaka University, Suita, Osaka 565–0871, Japan
[a)] s-salita@ist.osaka-u.ac.jp
[b)] takeuchi@ist.osaka-u.ac.jp
[c)] imai@ist.osaka-u.ac.jp

posed. These models can accelerate the simulation by the orders of magnitude, but they must be rebuilt for each of the architectures. On the other hand, the flexible bus model-based approaches for communication refinement [9], [10] repeat only the performance analysis steps for different bus architectures, but the remodeling and simulation are still needed to collect the communication trace for architectures that contains different processing element set. Yet, the overall estimation still takes too long.

The simulation of models using high-level languages, such as SpecC or SystemC, are faster than the conventional HDL modeling and simulation approach. These languages support system modeling and simulation in several abstraction levels. Loghi et al. proposed a Cycle Accurate (CA) model [11]. The simulation speed of a CA model is likely to be 10–100 times faster than the speed of a RTL simulation [12]. Models of AMBA shared bus protocols [13] and multi-layer bus protocols [14] are proposed to approximate system performance in the level of bus cycle accurate at transaction boundaries. AMBA shared buses are also modeled at transaction level to capture arbitration and bus contention [15]. Baganne et al. has shown that the simulation of Bus Cycle Accurate (BCA) model is approximately 19–90 times faster than the RTL simulation depending on the test data and the timed-model's simulation is about 20 times faster than the BCA's [16]. However, the speed of high-level simulations, e.g., simulating timed-model, BCA model, or CA model using system-level languages, is still slow and the performance estimation of each architecture requires an individual high-level abstraction model, which takes up to 3 and 4 days of modeling effort to create timed- and BCA model for each architecture [13], respectively. Although higher abstraction level models such as fast functional model allows simulation time speed-up between 20–110 times over CA model [17], they are mainly used for functional verification.

It is undeniable that an accurate simulation-based performance evaluation is necessary in the final design procedure, but in an early design stage which explores potential architectures, a quick estimation method is more crucial than the highly-accurate but slow ones. Most fast estimation methods employ a static Model of Computation (MoC). In Ref. [18], a formal model is used for approximating the performance of AMBA shared bus and detecting a deadlock. In Ref. [19], a Synchronous Data Flow (SDF) offers an analytic properties and architecture-specific overhead is analyzed for performance estimation of hierarchical shared bus. Rather than a worst-case timing analysis, a statistical performance analysis using a stochastic timed marked graph [20] and a timed marked graph [21] are proposed. The research of Cho et al. [22] comes closest to ours because it estimates system bus latency for both shared bus and multi-layer shared bus. Nonetheless, the analysis of these models fails to capture the dynamic bus contention during system execution, which is the main cause of estimation inaccuracy.

Ueda et al. proposes a performance estimation method based on system-level profiling [1], our preceding study. A system-level profiling is a simulation of a loosely-timed model, which is at least 20 times faster than BCA model's simulation. Their target architectural model includes functional blocks, instances of Intellectual Properties (IP), simple shared buses and buffers. The method simulates a user-defined system-level model to obtain execution order and the amount of transferred data from the profiling procedure, and constructs System-Level Execution Dependency Graph (SL-EDG) accordingly. Then, the Architecture-Level Execution Dependency Graph (AL-EDG) is constructed by adding edges representing dependencies raised by the availability of buffer resources and analyzed to estimate system execution time. Although the method is effective in terms of time spent for the estimation process, it has three main limitations. First, bus model is limited to shared bus and a data transfer must be conducted by only one bus. Consequently, each system-level channel must occupy a dedicated functional block's port that is connected to a bus. Second, the assumption regarding data communication does not satisfy master-slave communication concept, which exists in most high-speed bus protocols. Third, the performance analysis models neither dynamic bus behavior nor deadlock state. For that reason, probable bus operations are ignored and deadlock cannot be detected.

In this research, the multi-layer bus architecture and bus protocols are studied and modeled in order to efficiently estimate performance of SoC architecture. The architectural model is extended so that it can also represent configurable multi-layer bus, memory and Direct Memory Access (DMA) controller engaged in data communication. Communication port model is also improved to indicate master-slave roles of ports on the connecting bus and specify port sharing among multiple channels. In terms of AL-EDG construction procedure, instead of adding buffer-related dependency edges, our proposed method inserts additional vertices representing executions on memories, DMA controllers and related data transfers according to data communication path specified by the architectural model. In the analysis procedure, master or slave roles of communication ports and buffer status are also considered when analyzing bus requests in this study in addition to the execution dependencies considered in the preceding study. Furthermore, bus contention is recognized in order to predicts probable dynamic bus behavior, i.e., split, retry and preemption operation. With our proposed method, the performance of SoC can be evaluated quickly and accurately.

3. Definitions

First, this section explains MoC, architectural model of the configurable multi-layer bus-based SoC, and defines the proposed performance estimation method.

3.1 Model of Computation (MoC)

A Kahn-Process Network-based acyclic directed graph called System-Level Model (SLM) is used as our MoC to specify behavior of a target system in terms of sequential data computation processes and unbounded FIFO communication channels. An SLM is described as a loosely-timed model of the TLM 2.0 specification [23], in which processes expressing data processings are untimed, while the entry points and exit points of channels expressing data transfers are explicitly noted by event triggers.

An SLM is represented by $M_{sl} = (P, C)$, which means that SLM M_{sl} is composed of a process set $P = \{p_i | i = 0, 1, 2, \ldots\}$,

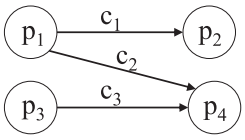

Fig. 1 An example of SLM.

and a channel set $C = \{c_j | j = 0, 1, 2, \ldots\}$. $c_j = (p_m, p_n)$ represents the channel from process $p_m \in P$ to $p_n \in P$. The data to be used in a process is received through the input channels, executed inside the process and the result is transmitted via the output channels. A write operation to an output channel is a non-blocking operation, while a read operation from an input channel is a blocking one. Additionally, s_j and p_{c_j} represent the data size and the execution priority of channel c_j, respectively.

Figure 1 shows an example of an SLM, M_{sl}, consisting of four processes and three channels. An arrow represents the direction of data flow in each channel. For instance, channel c_1, c_2 and c_3 are data communications from process p_1 to p_2, p_1 to p_4 and p_3 to p_4, respectively.

3.2 Architectural Model

A configurable multi-layer bus-based architecture consists of IP modules, DMA controllers, memories, shared buses and/or a multi-layer bus. A multi-layer bus is composed of a bus matrix and the buses on it, which allows the parallel communications in a system with multiple masters and slaves. An architecture may contain heterogeneous configurations of a multi-layer bus [24] such as multiple masters, multiple slaves, local slave and sub-systems. Some of the configurations degrade performance, but removing unnecessary buses on bus matrix and optimizing bus matrix with these configurations give a benefit in area and ease of routing.

The configurable multi-layer bus-based architectural platform is formalized as the architectural model called Architecture-Level Model (ALM). An ALM describes components and organization of an architecture, including the information about process-to-functional block and channel-to-port mapping decisions. One channel is accounted for the point that the data flows into it, called source of channel, and the point that the data flows out of it, called destination of channel, to be mapped on to the ports that are responsible for the transfers. An ALM is defined with a 7-tuple, (F, PT, D, M, B, BM, BB), as follows;

- F is a set of IP modules' instances, called functional blocks, that undertake the execution of the system-level processes. $fb_i = (j, P_{fb_i}, f_{fb_i}, e_{(p_k, fb_i)}) \in F$ indicates that functional block i is an implementation of IP j and undertakes the processes in set P_{fb_i}. f_{fb_i} and $e_{(p_k, fb_i)}$ represent the operation frequency of fb_i and execution cycle of process $p_k \in P_{fb_i}$ on functional block fb_i, respectively.
- PT is a set of ports $pt_i = (fb_j, b_k, n_q, n_r)$. A port connects a functional block fb_j to a shared bus b_k, and functions as a master or a slave on the connecting bus. A port contains n_q receive buffers and n_r transmit buffers for multiple buffering.
- D is a set of DMA controllers, $d_i = (C_{d_i})$. C_{d_i} refers to a set of source and destination of channels that requires d_i to initiate the transfer. A DMA controller functions as a bus master to transfer data from a requesting slave, store data in buffer temporarily, and send data to another slave upon request.
- M is a set of memories, $m_i = (C_{m_i}, n_{c_i})$. C_{m_i} refers to the set of source and destination of channels that are conducted via a memory, which serves as a slave in the system. The memory space is divided into storage blocks, and the number of storage blocks to store data of c_i is represented by n_{c_i}.
- B is a set of shared buses, $b_i = (w_{b_i}, f_{b_i}, pr_{b_i}, bmp_{b_i})$. w_{b_i}, f_{b_i} and pr_{b_i} are b_i's data bus width, frequency and protocol. bmp_{b_i} indicates the port of bus matrix that b_i is connected.
- BM represents multi-layer bus's bus matrix, which is defined with a 4-tuple, $(w_{bm}, f_{bm}, pr_{bm}, BBM)$, where w_{bm}, f_{bm} and pr_{bm} is data bus width, frequency and protocol, respectively. BBM is a set of buses, bbm_i, that route bus matrix's master layers to slave layers on the fabric of multi-layer bus.
- BB is a set of bus bridges. $bb_i = (b_j, b_k)$, represents the bus bridge that connects its master interface to b_j and its slave interface to b_k, implying that bb_i functions as a bus master on b_j and as a slave on b_k.

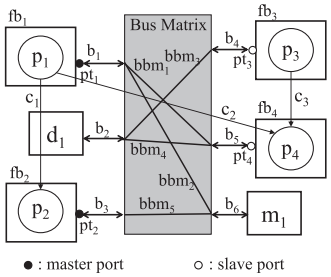

Fig. 2 An example of ALM.

Figure 2 shows an example of an ALM, M_{al}, composed of four functional blocks, four ports, a DMA controller, a memory, six shared buses and five buses on bus matrix of multi-layer bus. Processes and channels of M_{sl} in Fig. 1 are mapped onto components in M_{al}. The process-to-functional block mapping information specifies that p_1, p_2, p_3 and p_4 are mapped onto fb_1, fb_2, fb_3 and fb_4, respectively. Similarly, the channel-to-port mapping indicates that the sources of c_1 and c_2, symbolized with c_{1s} and c_{2s}, are mapped onto master port pt_1, the source of c_3, c_{3s}, is mapped onto slave port pt_3, the destination of c_1, c_{1d}, is mapped onto master port pt_2, while the destinations of c_2 and c_3, c_{2d} and c_{3d}, are mapped onto slave port pt_4.

From the channel-to-port mapping, C_{d_i} and C_{m_i}, the communication path of each channel is determined considering the master-slave communication regulation of bus protocols. In Fig. 2, channel c_2's communication path is "$pt_1 \rightarrow pt_4$." In the case of c_1, $C_{m_1} = \{c_{1s}\}$ because memory is necessary as an intermediate slave of the communication between two master ports. Therefore, the communication path of c_1 becomes "$pt_1 \rightarrow m_1 \rightarrow pt_2$," meaning that the data transfer is conducted from pt_1 to m_1 and from m_1 to pt_2. Likewise, $C_{d_1} = \{c_{3s}\}$ because a DMA controller is needed as a master in the communication of c_3 and the communication path becomes "$pt_3 \rightarrow d_1 \rightarrow pt_4$." A communication path may traverse more than one DMA controller or memory due to the placement of ports on the buses connected to the bus matrix. Besides, a transfer in a sub-path, e.g., "$pt_1 \rightarrow m_1$," may involve multiple buses and buses on bus matrix.

3.3 Definition of the Proposed Efficient Performance Estimation Method

- Input
  (1) M_{sl}: An SLM describing behavior of a system.
  (2) M_{al}: An ALM specifying components and mappings of an architecture.
- Output
  Total execution time of a system described by M_{sl} when executed on architecture M_{al}, considering concurrent data processings and transfers.

4. Performance Estimation Method for Configurable Multi-layer Bus-based SoC

There are four major procedures in the proposed efficient performance estimation method.

(1) System-level profiling - SLM is simulated in order to gather profiling information, which includes data processing timings, transfer timings and the amount of transferred data.
(2) SL-EDG construction - A graph representing execution dependencies in system-level between data processings and transfers is constructed from profiling information.
(3) AL-EDG construction - A graph representing architecture-dependent execution orders between data processings and transfers is constructed from SL-EDG and ALM.
(4) AL-EDG analysis - Performance of each ALM is estimated by analyzing corresponding AL-EDG to obtain the architecture-dependent data processing and transfer timings.

Since both time-consuming profiling procedure and SL-EDG construction procedure are architecture-independent, they are done only once for all ALMs of the same SLM and input data. Therefore, it is possible to quickly estimate the performance of various architectures by iteratively constructing and analyzing AL-EDG without simulating every individual architecture.

4.1 System-Level Profiling Using SystemC

In order to gather profiling information, monitoring process class and monitoring channel class are extended from SystemC's sc_module and sc_prim_channel, respectively, because SystemC can model hardware's parallel execution [23]. Each process of SLM is implemented with monitoring process class to capture timings of data processings. Likewise, each channel is implemented with monitoring channel class to monitor the amount of transferred data and data transfer timings, which are recorded when there are both read access and write access to the channel.

System-level profiling is proceeded by compiling SLM's code and executing its binary, meaning that it is done in a loosely-timed manner. Consequently, all profiling information is quickly gathered using SystemC simulator.

4.2 SL-EDG Construction

An SL-EDG is a graph that represents data processings, data transfers and their execution dependencies in system-level. It is constructed based on the profiling information. Its construction is independent of hardware architecture, so does the number of its vertices. SL-EDG is represented by $G_{sl} = (V_{sl}, E_{sl})$. The SL-EDG G_{sl} is comprised of the set of system-level vertices $V_{sl} = \{v_{p_{(i,k)}} \vee v_{c_{(j,l)}} \mid i, j, k, l \in \mathbb{N}\}$, and the set of system-level edges $E_{sl} = \{(v_{p_{(i,k)}}, v_{p_{(i,k+1)}}) \vee (v_{c_{(j,l)}}, v_{c_{(j,l+1)}}) \vee (v_{p_{(i,k)}}, v_{c_{(j,l)}}) \vee (v_{c_{(j,l)}}, v_{p_{(i,k)}}) \mid i, j, k, l \in \mathbb{N}\}$.

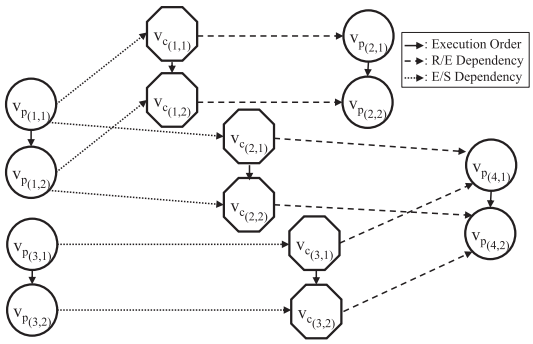

Fig. 3 An example of SL-EDG.

Figure 3 illustrates the example of SL-EDG corresponding to SLM in Fig. 1. Assuming that each process executes its data processing twice and each channel transfers data twice. The circular nodes denoted by $v_{p_{(i,k)}}$ represents the vertex of p_i's k-th data processing and the octagonal nodes denoted by $v_{c_{(j,l)}}$ represents the vertex of c_j's l-th data transfer. The solid arrows represent execution orders. The dashed arrows represent R/E dependency-edges, indicating that the execution of data processing starts after the data has been received, and the dotted arrows represents E/S dependency-edges, indicating that the data transmission starts after the execution of data processing is over.

More details about system-level profiling and SL-EDG construction are mentioned in another literature [1].

4.3 AL-EDG Construction

An AL-EDG is a graph that represents data processings, data transfers and their execution dependencies according to components of the architecture specified by an ALM. In addition to the vertices and edges of SL-EDG, AL-EDG also consists of vertices and edges involving DMA controllers and memories that fulfill bus protocol's regulation about master-slave communication of each data transfer. The number of its vertices depends on the components and organization of the ALM. AL-EDG is represented by $G_{al} = (V_{al}, E_{al})$, where V_{al} and E_{al} are AL-EDG's vertex set and edge set, respectively.

The AL-EDG is constructed by the following steps;

(1) Copy SL-EDG as AL-EDG. Let V_{al} be V_{sl} and E_{al} be E_{sl}.
(2) Alter V_{al} and E_{al} so that the AL-EDG also includes the dependencies of data transfers raised by communication paths. For every channel $c_i = (p_u, p_x) \in C$, do as follows;
  (a) If $c_{is} \in C_{d_k}$, meaning that DMA controller d_k initiated c_i's transfer to a port mapped to the source of channel c_i, do as follows;
    (i) Make vertices $v_{d_{(k,l)}}$ representing processings on d_k, and vertices $v_{c'''_{(i,j)}}$ representing additional data transfers of c_i. Then, add them to V_{al}. Make edges $(v_{c'''_{(i,j)}}, v_{c'''_{(i,j+1)}})$ representing execution orders between data transfers of c_i, and add them to E_{al}.
    (ii) Delete edges $(v_{c_{(i,j)}}, v_{p_{(x,y)}})$ from E_{al} and add edges $(v_{c_{(i,j)}}, v_{d_{(k,l)}})$, $(v_{d_{(k,l)}}, v_{c'''_{(i,j)}})$ and $(v_{c'''_{(i,j)}}, v_{p_{(x,y)}})$, which represent execution dependencies in c_i's commu-

nication path traversing d_k, to E_{al}.

(b) If $c_{is} \notin C_{d_k}$ and $c_{is} \in C_{m_q}$, meaning that a port mapped to the source of channel c_i establishes c_i's transfer to memory m_q, do as follows;

(i) Make vertices $v_{m_{(q,r)}}$ representing processings on m_q, and vertices $v_{c'''_{(i,j)}}$ representing additional data transfers of c_i. Then, add them to V_{al}. Make edges $(v_{c'''_{(i,j)}}, v_{c'''_{(i,j+1)}})$ representing execution orders between data transfers of c_i, and add them to E_{al}.

(ii) Delete edges $(v_{c_{(i,j)}}, v_{p_{(x,y)}})$ from E_{al} and add edges $(v_{c_{(i,j)}}, v_{m_{(q,r)}})$, $(v_{m_{(q,r)}}, v_{c'''_{(i,j)}})$ and $(v_{c'''_{(i,j)}}, v_{p_{(x,y)}})$, which represent execution dependencies in c_i's communication path traversing m_q, to E_{al}.

(c) If $c_{is} \in C_{d_k}$ and $c_{is} \in C_{m_q}$, meaning that the communication of c_i traverses memory m_q after DMA controller d_k, do as follows;

(i) Make vertices $v_{m_{(q,r)}}$ representing processings on m_q, and vertices $v_{c'_{(i,j)}}$ representing additional data transfers of c_i. Then, add them to V_{al}. Make edges $(v_{c'_{(i,j)}}, v_{c'_{(i,j+1)}})$ representing execution orders between data transfers of c_i, and add them to E_{al}.

(ii) Delete edges $(v_{d_{(k,l)}}, v'''_{c_{(i,j)}})$ from E_{al} and add edges $(v_{d_{(k,l)}}, v'_{c(i,j)})$, $(v'_{c_{(i,j)}}, v_{m_{(q,r)}})$ and $(v_{m_{(q,r)}}, v_{c'''_{(i,j)}})$, which represent execution dependencies in c_i's communication path traversing m_q after d_k, to E_{al}.

(d) If $c_{id} \in C_{d_s}$, meaning that the transfer of c_i traverses DMA controller d_s after memory m_q, do as follows;

(i) Make vertices $v_{d_{(s,t)}}$ representing processings on d_s, and vertices $v_{c''_{(i,j)}}$ representing additional data transfers of c_i. Then, add them to V_{al}. Make edges $(v_{c''_{(i,j)}}, v_{c''_{(i,j+1)}})$ representing execution orders between data transfers of c_i, and add them to E_{al}.

(ii) Delete edges $(v_{m_{(q,r)}}, v'''_{c_{(i,j)}})$ from E_{al} and add edges $(v_{m_{(q,r)}}, v''_{c(i,j)})$, $(v''_{c_{(i,j)}}, v_{d_{(s,t)}})$ and $(v_{d_{(s,t)}}, v_{c'''_{(i,j)}})$, which represent execution dependencies in c_i's communication path traversing d_s after m_q, to E_{al}.

(3) Divide the vertices into groups of functional blocks V_{fb_i}, buses V_{b_i}, buses on bus matrix V_{bbm_i}, DMA controllers V_{d_i}, and memories V_{m_i}, that undertake their executions. The channel vertices must be included in the groups of all buses undertaking their executions.

In the following, the AL-EDG shown in **Fig. 4** is constructed for the ALM shown in Fig. 2. First, the SL-EDG in Fig. 3 is copied as an initial AL-EDG. Since $C_{d_1} = \{c_{3s}\}$, $v_{d_{(1,1)}}$, $v_{d_{(1,2)}}$, $v_{c'''_{(3,1)}}$ and $v_{c'''_{(3,2)}}$ are generated into the graph in step 2(a)i. Then, in step 2(a)ii, edges $(v_{c_{(3,1)}}, v_{p_{(2,1)}})$ and $(v_{c_{(3,2)}}, v_{p_{(2,2)}})$ are removed, and edges $(v_{c_{(3,1)}}, v_{d_{(1,1)}})$, $(v_{c_{(3,2)}}, v_{d_{(1,2)}})$, $(v_{d_{(1,1)}}, v_{c'''_{(3,1)}})$, $(v_{d_{(1,2)}}, v_{c'''_{(3,2)}})$, $(v_{c'''_{(3,1)}}, v_{p_{(2,1)}})$ and $(v_{c'''_{(3,2)}}, v_{p_{(2,2)}})$ are added to the AL-EDG. Similarly, because $C_{m_1} = \{c_{1s}\}$, the graph is modified according to steps 2(b)i and 2(b)ii as marked. Finally, the vertices are grouped. $v_{p_{(1,1)}}$ and $v_{p_{(1,2)}}$ are put into V_{fb_1}, the group of process vertices undertaken by fb_1. Similarly, $v_{d_{(1,1)}}$ and $v_{d_{(1,2)}}$ are grouped into V_{d_1}, the group of DMA controller vertices undertaken by d_1. V_{b_2}, the group of channel vertices undertaken by b_2, includes vertices $v_{c_{(3,1)}}$, $v_{c_{(3,2)}}$, $v_{c'''_{(3,1)}}$ and $v_{c'''_{(3,2)}}$. Likewise, V_{b_5} includes vertices $v_{c_{(2,1)}}$, $v_{c_{(2,2)}}$, $v_{c'''_{(3,1)}}$ and $v_{c'''_{(3,2)}}$. In the example, $v_{c'''_{(3,1)}}$ and $v_{c'''_{(3,2)}}$, also in V_{bbm_4},

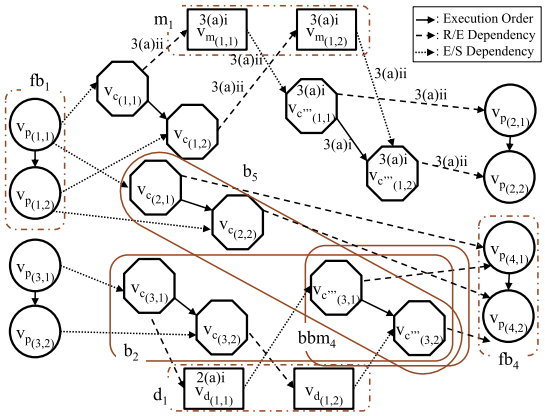

Fig. 4 An example of AL-EDG.

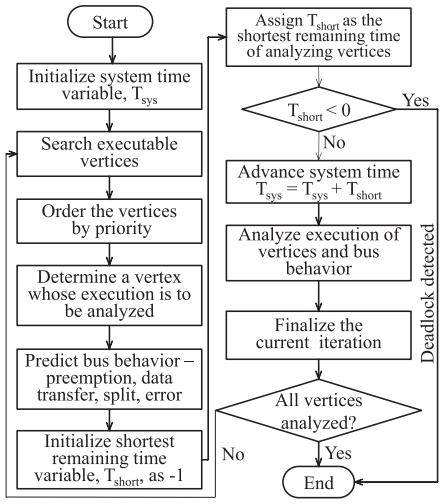

Fig. 5 The flow of AL-EDG analysis.

the group of bbm_4, are included in three groups because the transfer from DMA controller to fb_4 uses b_2, b_5 and bbm_4.

4.4 AL-EDG Analysis

AL-EDG analysis estimates the performance by predicting the behavior of the target system on a specified architecture. This paper aims to speculate multi-layer and shared bus behavior based on the analyzed dynamic bus contention arising from arbitration, traffic congestion, advanced bus features and communication protocols. Therefore, the general concepts of bus protocols and advanced bus features of both multi-layer bus and shared bus are modeled so that the AL-EDG analysis recognizes dynamic bus contention. As a result, the proposed method earns an advantage in terms of accuracy and performance estimation speed.

The AL-EDG analysis flow, shown in **Fig. 5**, begins with system time variable initialization. Then, the analysis steps are iterated to execute data processings and transfers until every vertex in V_{al} exhausts or the deadlock is detected.

First, the analysis finds the executable vertices of the current iteration. A vertex in V_{fb_i}, V_{d_j} and V_{m_k} is classified as executable when it has no edge from other vertices, and added to executable vertex set $V_{exe_{fb_i}}$, $V_{exe_{d_j}}$ and $V_{exe_{m_k}}$, respectively. In order to model the regulation of bus protocol that bus master initiates a communication to slave, a channel vertex is considered as executable based on the master's request. That is, a channel vertex whose transfer

is initiated by a master port becomes executable when the status of one of the its receive buffers is empty for a read transaction or the vertex has no edge from other vertices for a write transaction. A channel vertex whose transfer is initiated by a DMA controller is executable when it has no edge from other vertices and for a write transfer, the status of one of the slave port's receive buffers is empty. Then, the vertices are added to executable vertex set $V_{exe_{b_i}}$ and $V_{exe_{bbm_j}}$. The existence of multiple vertices within an executable vertex set of bus or bus on bus matrix implies simultaneous bus requests and activities. For that reason, together with the current bus activity, bus contention is detected.

In the next step, the vertices in each executable vertex set are reordered by priority, which is decided according to scheduling and arbitration policy at analysis time. Priorities of process vertices are determined from user-defined process priorities on each functional block. Priorities of channel vertices in $V_{exe_{b_i}}$ of the shared buses on the master layer side of bus matrix are determined based on the priorities of bus master and bus bridge's master interface designated by system designer, so do priorities of those in $V_{exe_{bbm_j}}$. On the other hand, priorities of those in $V_{exe_{b_i}}$ of the bus connected to the slave layer of bus matrix depend on priorities of the bus matrix's master layer.

Then, a vertex whose execution will be analyzed is selected from each executable vertex set. The analysis program selects the process vertex that has the highest priority and the status of one of the target port's transmit buffers is empty from $V_{exe_{fb_i}}$, and the channel vertex that has the highest priority and the master is not banned by bus's arbiter from $V_{exe_{b_i}}$ and $V_{exe_{bbm_j}}$. However, some channel vertices in $V_{exe_{b_i}}$ and $V_{exe_{bbm_j}}$ depend on the selected vertices of other $V_{exe_{b_i}}$ and $V_{exe_{bbm_j}}$ whether it can be selected. For instance, the channel vertex whose transfer involves bus bridge $bb_q = (b_r, b_s)$ can be selected from $V_{exe_{b_r}}$ only when it holds the highest priority among the vertices in $V_{exe_{b_s}}$.

The analysis predicts dynamic behavior of bus architecture from the selected channel vertices and the speculated bus contention. Bus activity is determined when a channel vertex is selected from every executable vertex set of buses and bus on bus matrix that it belongs to. Split or retry response's operation, the mechanism that allow the shared bus and bus on bus matrix to be released when the slave cannot conduct normal data transfer immediately, is diagnosed when the status of slave's receive buffer is not empty for write operation or there exists incoming edges to the channel vertex for read operation. In the analysis of such cases, the slave is assumed to response with retry when the transfer traverses bus on bus matrix, and with split otherwise. Bus preemption is detected if the selected vertex of a bus's executable set holds a higher priority than the one analyzed as executing on the bus and the transfer on the bus is not analyzed as a lock transfer. Otherwise, the occurrence of normal data transfer is determined.

Next, after initializing the shortest remaining time variable, T_{short}, as -1, the remaining operation time of each analyzing vertex is computed by deducting elapsed time from estimated total operation time of the vertex and bus activity, and the shortest time is assigned to T_{short}. The system time is advanced by the time assigned to T_{short}. However, since there is a chance that the analyzing system falls in a deadlock state, the analysis program detects the deadlock if T_{short} is less than 0, terminates immediately and reports the deadlock condition to the user. In this case, the user may modify the channel priority description and run the analysis again in order to resolve the deadlock.

Total data processing time of each process vertex is calculated in advance by the following equation;

$$t_p = \frac{e_{(p_i, fb_j)}}{f_{fb_j}} \tag{1}$$

Total processing time of DMA controller and memory vertices are assumed to be 0, since both components only temporarily store data in their internal storage.

On the other hand, total bus usage time is determined based on the predicted bus activity in every iteration so that dynamic bus contention effects are recognized. The time for split operation is determined as in Eq. (2).

$$t_s = \frac{S + C_c + C_a}{\min(f_{b_i})} \tag{2}$$

S and C_c are the overhead of split operation and protocol conversion, respectively, while C_a is the number of address cycles and $\min(f_{b_i})$ represents the lowest bus frequency among the frequency of buses that the split operation takes place. The time for retry operation is determined similarly as in Eq. (3), where R denotes the overhead of retry operation.

$$t_r = \frac{R + C_c + C_a}{\min(f_{b_i})} \tag{3}$$

Finally, the calculation of data transfer time is shown in Eq. (4),

$$t_d = \frac{D \times C_d \times B + C_c + C_a}{\min(f_{b_i})} \tag{4}$$

$$D = \frac{w_{c_i}}{\min(w_{b_i})} \tag{5}$$

where C_d and B are the number of clock cycles in one data cycle and the number of burst beats, respectively. D is the number of data cycles required to transfer one data, determined by Eq. (5), where w_{c_i} is the number of bits of one data transferred by the analyzing channel vertex and w_{b_i} represents the bit width of the narrowest data bus among the buses and bus matrix that the transfer takes place. The number of address cycles implies pipeline nature of the bus. It is counted as 0 if the new data transfer is consecutive to the previous one or as the number of protocol's address cycles, otherwise. The number of burst beats is determined according to the number of remaining data to be analyzed and bus preemption.

By analyzing the selected vertices and the speculated bus behavior, the analysis program advances elapsed time of the analyzing vertices and bus activities by T_{short}, and keeps track of system's resource status. For the process and channel vertices analyzed for the first time, the IDs of the vertices are registered to models of the target storage to track the status of buffers in ports, DMA controllers and storage blocks in memories. An ID is unregistered from the storage model when the dependent vertices are analyzed as completed, implying that the data is processed or transferred. The storage model with no ID registered implies that the status is empty. Since data transfer of a channel vertex might be separated into several burst transfer's analysis, the number of remaining transfer data is deducted by the number of

burst beats, B, when elapsed time becomes equal to total operation time. The vertices are recognized as completed when their elapsed time becomes equal to the total operation time except for the channel vertices that the number of remaining data must also become 0. In each channel vertex's analysis, bus bridge in use is marked as active. Shared bus and bus on bus matrix resources are marked as active, lock, split and retry to represents the data transfer, lock transfer, split and retry operation on the bus, respectively. Additionally, the bus master whose transfer is split is marked as banned from arbitration. Lastly, the completed vertices and related edges are removed from the G_{al}.

Finally, the status of each resource is finalized according to the bus protocol at the end of each iteration, e.g., unban bus master, etc. If there are no vertices left in the G_{al}, the analysis returns T_{sys} as system time. Otherwise, the analysis loops from searching the executable vertices step.

4.5 Computational Complexity

The computational complexity of the AL-EDG analysis is derived from the flow explained previously and the program implemented to estimate the performance of multi-layer AHB bus-based SoC, which is described thoroughly in Section 5.

The asymptotic notation $O(n^3)$ expresses the proposed AL-EDG analysis's computational complexity as a function of the number of AL-EDG vertices, n. The analysis repeats from searching executable vertices to finalizing iteration for at most kn iterations, where k is a constant indicating the number of loops spent for analyzing a vertex. In each iteration, the most complex step in the computational time aspect is ordering the executable vertices, where the worst case consumes n^2 time complexity. Therefore, the complexity of the proposed analysis in the worst-case is cubic w.r.t. the number of AL-EDG vertices.

However, the computational complexity becomes $O(n^2)$ in most cases that the processes execute iteratively and each of the components in an architecture undertakes only a few number of processes and channels. Consequently, a few executable vertices exist in the executable vertex sets at a time and ordering the executable vertices consumes only n order of time complexity. The $O(n^2)$ complexity of most cases is depicted in Section 5.5.

The complexity applies to the situation that the application size, e.g., the size of image in the image processing, causes both the number of vertices in SL-EDG and AL-EDG to grow. In other words, AL-EDG analysis runtime increases by $O(n^3)$ when estimating the performance of various-sized applications executed on the same architecture.

5. Case Study

To show that proposed method is efficient to be included in architecture exploration of ESL, the proposed analysis flow is applied for performance analysis of multi-layer AHB bus-based SoC. The proposed method is also applicable to shared bus-based architecture and not limited to AHB bus protocol.

The efficiency of the proposed performance estimation method is investigated in two aspects. The first one is the accuracy of the performance estimated by the proposed method when compared with the performance obtained from the RTL simulation. The second one is the speed-up of the proposed method over the RTL simulation, which is measured from the runtime of both tools.

Table 1 List of protocol's parameters.

Parameter	Value
S	2
R	2

Table 2 List of protocol related variable values.

Variable	Value
C_d	1, 2
C_c	0, 1
C_a	0, 1
B	1, 2, 4, 8, 16

5.1 Modeling of Multi-layer AHB Protocol

In order to apply the proposed flow to analyze the performance of a multi-layer AHB bus-based system, some protocol parameters, protocol related variable values, and conditions are specified.

Protocol's parameters and protocol related variable values are defined based on AHB protocol of the AMBA specification [25] as shown in **Tables 1** and **2**. The split and retry responses of AHB requires two cycles as the overhead, therefore, both overhead of split operation S and overhead of retry operation R are set to 2. The values for protocol related variables are determined according to system status during the analysis, but restricted to a certain set of values. The number of clock cycles in one AHB and APB data cycle C_d is 1 and 2 under the assumption that there is no wait cycle. The overhead of protocol conversion differs by protocol pairs and direction of data flow. For AHB-APB protocol conversion, C_C is 1 for a write transfer and 0 for a read transfer, split response and retry response. The number of address cycles C_a can be either 0 when pipelined, otherwise the AHB address cycles with no wait state is 1. The number of burst beats B is typically 1, 2, 4, 8 and 16, except only when bus preemption occurs that B can be an integer not more than 16.

There are three additional multi-layer AHB protocol conditions. Firstly, every communication via bus matrix must be locked because the arbiters of multi-layer bus do not allow preemption. Consequently, the shared buses used for the transfer must be marked as lock. The second condition regards the state machine of AHB master interface that 1-cycle-idle phase presents between two bus requests. Therefore, a channel vertex is removed from executable vertex sets for one clock after the analysis of the previous operation has finished. Finally, the arbitration policy of buses and buses on bus matrix is restricted to fixed-priority policy.

The following describes the analysis of the G_{al} in Fig. 4. Let the priorities of fb_1, fb_2 and d_1 be 3, 2, 1, respectively, so do the priorities of c_1, c_2 and c_3. Assume that the mapped functional blocks of p_1, p_2, p_3 and p_4 spend 100, 80, 120 and 140 ns for data processing calculated by Eq. (1) and the amount of data transferred in c_1, c_2 and c_3 are 16, 16 and 32, respectively. The system operates at 50 MHz, and there is one receive and one transmit buffer in each port. The execution Gantt chart is shown in **Fig. 6**.

After T_{sys} is initialized, the executable vertices are searched throughout G_{al}. In the first iteration, $v_{p_{(1,1)}}$ and $v_{p_{(3,1)}}$ have no source edge, so they are added to $V_{exe_{fb_1}}$ and $V_{exe_{fb_3}}$, respectively. Moreover, since pt_2 has an empty receive buffer, $v'''_{c_{(1,1)}}$ is analyzed

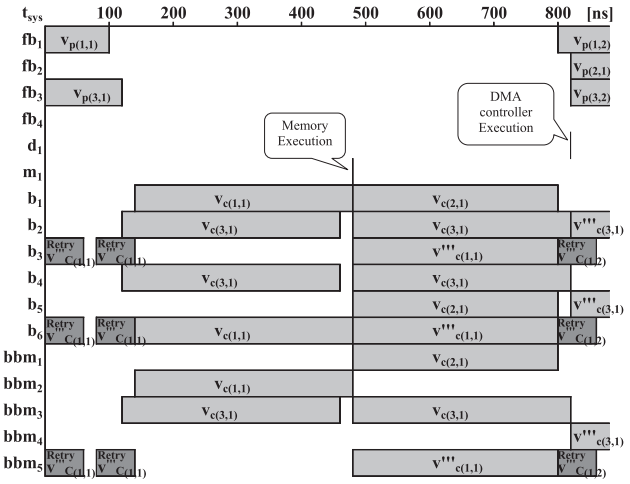

Fig. 6 An example of AL-EDG analysis.

Fig. 7 An SLM of JPEG encoder.

Table 3 Information of data in channels.

	width [bit]	#data
c_0	24	64
c_1	8	64
c_2	12	64
c_3	12	64
c_4	12	64
c_5	8	256

Table 4 Information of functional blocks and its ports.

FB name[*1]	Exe.cycle [cycle]	Port
BS	67	1 Slave
CT	68	1 Master
DCT	368	1 Slave
ZZ	67	1 Slave
Q	68	1 Master
VLC	200–265	1 Master
WRT	258	1 Slave

to have a bus request raised, and is added to $V_{exe_{b_3}}$, $V_{exe_{b_6}}$ and $V_{exe_{bbm_5}}$. Then, $v_{p_{(1,1)}}$, $v_{p_{(3,1)}}$ and $v'''_{c_{(1,1)}}$ are selected to be analyzed.

Next, the analysis predicts the bus activity of b_3, b_6 and bbm_5 to be retry response because there exists an edge to $v'''_{c_{(1,1)}}$.

Then, the shortest remaining time of analyzing vertices is determined. The remaining time of $v_{p_{(1,1)}}$ and $v_{p_{(3,1)}}$ are 100 and 120 ns, respectively, while the retry response of $v'''_{c_{(1,1)}}$ takes 60 ns according to Eq. (3) computed with one address cycle. For that reason, T_{short} becomes 60 and T_{sys} is advanced.

The vertices and bus behavior are analyzed. The ID of $v_{p_{(1,1)}}$ and $v_{p_{(3,1)}}$ are registered in the model of pt_1's and pt_3's transmit buffer. At the same time, the retry response finishes.

To finalize this iteration, $v'''_{c_{(1,1)}}$ is excluded from the analysis for 1 cycle due to AHB interface's idle phase. Consequently, T_{sys} is advanced by 20 ns in the second iteration.

In the third iteration, $v'''_{c_{(1,1)}}$ is reconsidered and analyzed to be responded with retry. T_{short} becomes 20 ns because the analysis of $v_{p_{1,1}}$ has ended, therefore, vertex $v_{p_{(1,1)}}$, edges ($v_{p_{(1,1)}}$,$v_{c_{(1,1)}}$) and ($v_{p_{(1,1)}}$,$v_{c_{(2,1)}}$) are removed from G_{al}.

In the fourth iteration, $v_{c_{(1,1)}}$ becomes executable and is added into $V_{exe_{b_1}}$, $V_{exe_{b_6}}$ and $V_{exe_{bbm_2}}$, so does $v_{c_{(2,1)}}$ which is added into $V_{exe_{b_1}}$, $V_{exe_{b_5}}$ and $V_{exe_{bbm_1}}$. Since both channel vertices are initiated from the same port, but c_1 holds a higher priority, $v_{c_{(1,1)}}$ is selected on b_1, b_6 and bbm_2, while $v_{c_{(2,1)}}$ is selected on only b_5 and bbm_1. Consequently, $v_{c_{(2,1)}}$ is ignored in this iteration. Unfortunately, because b_6 is occupied with retry response operation, $v_{c_{(1,1)}}$ is not analyzed. T_{sys} is advanced to 120 ns, and vertex $v_{p_{(3,1)}}$ and edges ($v_{p_{(3,1)}}$,$v_{c_{(3,1)}}$) are removed from G_{al}.

Then, $v_{c_{(3,1)}}$ is analyzed to transfer data for 16 burst beats, which takes 340 ns for one address cycle and 16 data cycles in the fifth iteration. However, the second retry operation of $v_{c_{(1,1)}}$ remains only 20 ns, so T_{sys} becomes 140 ns.

In the sixth iteration, The analysis of $v_{c_{(1,1)}}$'s transfer starts and lasts for 340 ns. At $T_{sys} = 460$, transfer of the first 16 data of $v_{c_{(3,1)}}$ finishes, but since there are 16 data left to be transferred, the vertex has to be considered again after 20 ns of 1-clock-cycle idle phase. At the same time, the analysis of $v_{c_{(1,1)}}$ remains 20 ns too, so T_{short} becomes 20 ns.

At $T_{sys} = 480$, the seventh iteration takes place to analyze the execution of memory vertex $v_{m_{(1,1)}}$. T_{short} equals 0 ns and the eight iteration begins at the same point of time. The executable vertex $v_{c_{(2,1)}}$ is selected to be analyzed and its execution time is 320 ns because the address cycle overlaps with the last data cycle of $v_{c_{(1,1)}}$. Meanwhile, $v_{c_{(3,1)}}$ is analyzed again and the transfer time becomes 340 ns.

The analysis proceeds in the same fashion until G_{al} exhausts.

5.2 Experimental Environment Setup

The effectiveness of the proposed performance estimation method is studied through a JPEG encoder application. **Figure 7** shows an SLM of JPEG encoder, consisting of seven processes and six channels. The processes are Block Splitting (BS), Color Transformation (CT), Discrete Cosine Transformation (DCT), Quantization (Q), ZigZag ordering (ZZ), Variable Length Coding (VLC) and file WRiTing (WRT). The data width and the amount of data in one system-level transaction of each channel are shown in **Table 3**. The images used in the experiments are processed in a 8×8 pixel block unit and without downsampling.

The experiments were conducted on a 3.60 GHz Intel Xeon, 32 GB memory and 64 bit CentOS5 machine. The estimation method is implemented in C language and the SLM was implemented with SystemC 2.3.0 [23]. The source code of the proposed method and the SLM was compiled with gnu gcc 4.1.2. RTL simulation tool is ModelSim SE-64 10.3.

5.3 Accuracy Measurement

The results of the AL-EDG analysis for performance estimation method are compared with the performance results of RTL simulation to measure the accuracy of the analysis. The performance of eight architectures, each of which executes the same aforementioned seven processes of JPEG encoder, are evaluated. The execution cycle shown in **Table 4** is the worst-case execu-

[*1] FB stands for functional block. The functional block executes the process with the same name.

Table 5 The number of vertices in SL-EDG and AL-EDG.

Image size	256 × 512	512 × 512	512 × 1,024	1,024 × 1,024
SL-EDG	59,542	118,972	238,234	476,628
AL-EDG	84,118	168,124	336,538	673,236

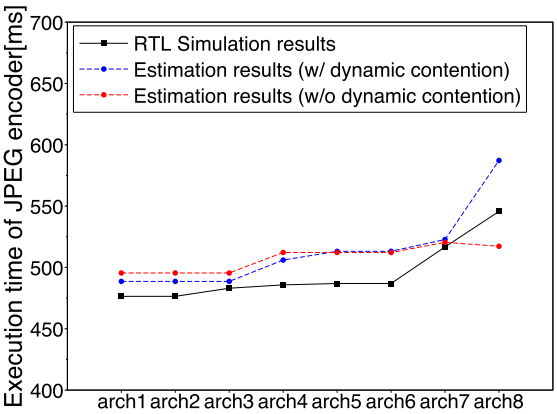

Fig. 8 Performance results from the proposed method, the method w/o considering dynamic bus contention and RTL simulation (1,024 × 1,024 pixel image)

tion cycle. Moreover, each architecture is comprised of the same set of functional blocks and ports, whose information is shown in Table 4, a DMA controller and a memory. The DMA controller functions as a master to initiate the communication between DCT and ZZ functional block. The memory stores data transferring between Q and VLC functional block. The width of the buses, shared bus and bus matrix, is 32 bits and each architecture is operated at 50 MHz. The architectures contain the same functional blocks, DMA controllers and memories, but their ports are attached to the bus architecture differently and various configurable multi-layer bus architectures are represented. Therefore, the difference in performance of the architectures are solely affected by the communication architecture.

The performance estimation by the proposed method is proceeded as follows; First, the encoding of an image was profiled to collect the data processing timing, data transfer timing and the amount of transferred data, and the SL-EDG was constructed accordingly. For each ALM of the architectures under evaluation, an AL-EDG was constructed and analyzed to obtain the estimated performance. **Table 5** shows the number of vertices in the SL-EDGs and AL-EDGs. Normally, the number of vertices in an AL-EDG depends on the components and communication path of every channel, but the number of vertices of AL-EDGs in the experiments are equal because the only difference is the architecture organization. However, the vertices are divided in to the groups of components that undertake them differently. The experiments were conducted with four images of different sizes.

Figure 8 shows the performance results yielded from RTL simulation, the proposed method considering dynamic bus contention and the method that does not consider dynamic bus contention. The results of the proposed method considering dynamic bus contention is represented in the graph as estimation results (w/ dynamic contention) and those of the method that does not consider dynamic bus contention is represented as estimation results (w/o dynamic contention).

The performance results of the proposed method considering dynamic bus contention are compared with the results of RTL simulation in order to evaluate the estimation error. Architectures noted as arch3 and arch7, respectively illustrated in **Figs. 9** (c) and (g), contain multi-layer bus with heterogeneous configurations and incur the smallest error of 1.5%. On the other hands, arch8, illustrated in Fig. 9 (h), is the shared bus-based architecture and incurs the biggest error of 7.6%. When bus contention is detected to occur on a shared bus, the analysis program speculates a probable operation as well as the operation time, which raises communication-related timing errors of each bus. These errors are accumulated especially when bus contention arises repeatedly. For that reason, the accuracy of the proposed method becomes worse in the case of a single shared bus-based architecture, arch8. In the case of multi-layer bus-based architecture, the errors are distributed to several shared buses and ensue on the estimated system time parallelly. According to the timing results, a larger amount of bus contention is found when analyzing performance of arch6, arch5 and arch4, respectively, so the errors of these architecture are bigger than the other architectures that contains multi-layer bus.

Figure 10 illustrates the mean values and the ranges of errors from comparing the performance results of eight architectures. The proposed method evaluates system performance with approximately only 1–8% difference from the RTL simulation and the mean values of error appears as 3.8%. Its overestimation is due to the fact that the Worst-Case Execution Time (WCET) of VLC functional block is constantly used in the estimation. On the other hand, the execution cycle of VLC varies because its execution behavior depends on the processed data.

During the performance analysis, taking dynamic bus contention, i.e., bus requests and current bus activity, and dynamic bus behavior, i.e., dynamic address phase calculation, split, retry and preemption operation, into account benefits the estimation results in many aspects. Figure 8 also illustrated the estimation results when neither dynamic bus contention nor probable dynamic bus behavior is recognized. They are labeled as estimation results (w/o dynamic contention). The first benefit is that analyzing system performance without considering dynamic bus contention and behavior may cause underestimation as of the one of arch8 because it assumes too optimistic bus contention. On the contrary, the address phase is not recognized dynamically when ignoring bus requests and current activity, so more errors incur in the estimation results of arch1, arch2, arch3 and arch4. This leads to a wider error range of −5.2% to 5.2%, which reduces the reliability of the estimation, and insufficient design, which is unacceptable because it might not satisfy the design constraints. The second benefit is that the proposed method's estimation results are approximately 1–2% more accurate than the results when bus contention and behavior are ignored. For instance, the estimation errors of arch1 from the method with and without the consideration regarding dynamic contention are 2.5% and 4.0%, respectively. However, since data processings of JPEG encoder application dominates data communications between IPs, the impact of considering dynamic bus contention and behavior is not so large.

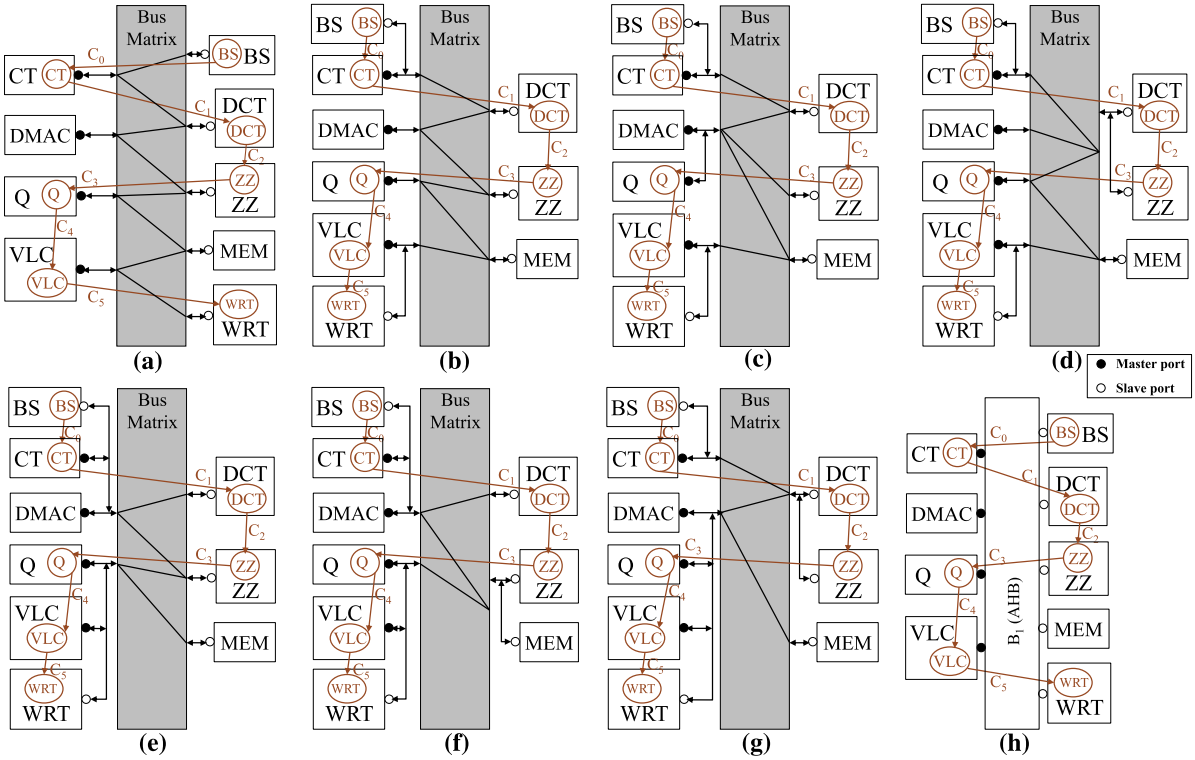

Fig. 9 ALM of architectures in the experiments. (a) arch1, (b) arch2, (c) arch3, (d) arch4, (e) arch5, (f) arch6, (g) arch7, (h) arch8.

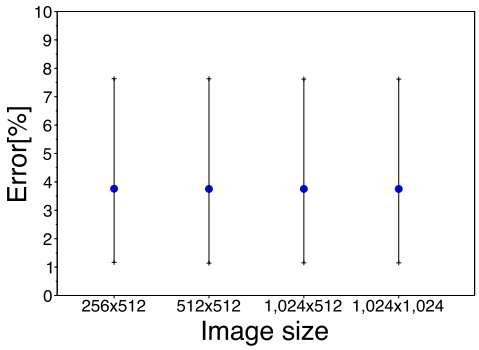

Fig. 10 Error bar shows the error of the estimation.

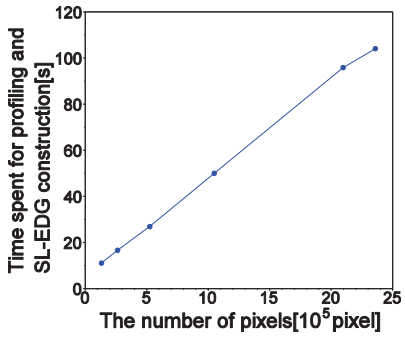

Fig. 11 Runtime for profiling and construction of SL-EDG.

5.4 Tool Runtime and Speed-Up

Figure 11 illustrates the time spent for system-level profiling and SL-EDG construction w.r.t. the number of pixels in the sample images. The procedures are done as fast as within two minute for the image as large as total 2,359,296 pixels (1,536 × 1,536 pixel) because the profiling is conducted in a loosely-timed TLM manner. However, the time spent for the two procedures tends to grow linearly for the bigger size of image.

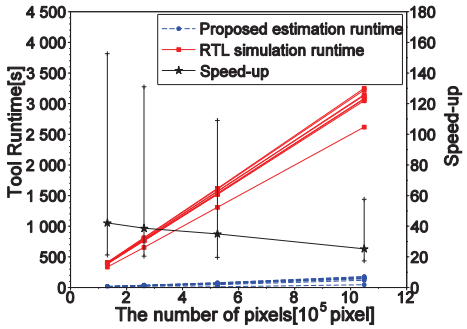

Fig. 12 Average speed-up in estimating performance of eight architectures.

The AL-EDG construction and analysis achieved much faster in evaluating the performance of an individual architecture, which is proven by the runtime speed-up value as high as 152.6 times over the simulation. **Figure 12** shows the relationship between the runtime of the proposed estimation method (circles)/RTL simulation (squares) of each architecture candidates and the number of pixels. The stars show the average speed-up of the architecture w.r.t. the number of pixels, and the relevant error bar indicates the range of speed-up values. The experiments demonstrate that the speed-up varies from 17.4–152.6 times by the combination and organization of functional blocks, DMA controller, memories and buses as well as the size of the sample image. The bigger the image is, the less the average speed-up becomes. This is because the runtime of the analysis program grows by a polynomial function as the image becomes bigger, while the RTL simulation runtime grows linearly in our case study.

Figure 13 draws the relationship between tools' runtime and

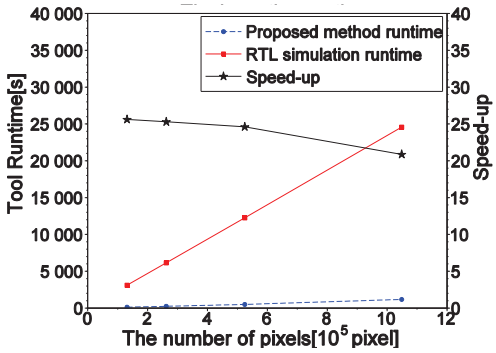

Fig. 13 The proposed method's overall speed-up.

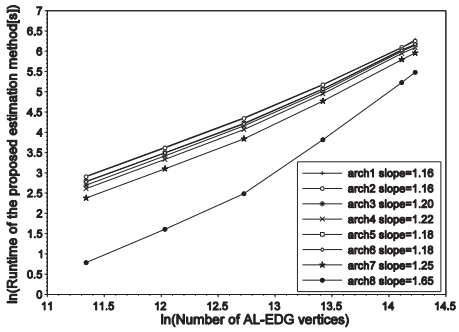

Fig. 14 AL-EDG analysis' runtime of individual architecture.

the number of pixels. It shows that the overall procedure of the efficient performance estimation method has achieved the maximum speed-up of 25.6 times over the overall RTL simulation in evaluating the performance of eight architectures. A bigger speed-up value can be gained when evaluating a larger number of architectures due to the fact that the proposed method conduct the profiling and SL-EDG construction procedure only once for one image. On the other hands, the speed-up slightly drops when approximating the performance of architectures encoding the bigger image. However, the proposed method is able to evaluate the performance of a large number of architectures within a much shorter time that the RTL simulation which takes unbearable long time.

5.5 Discussion

The abstraction level of the proposed method is between untimed- and timed-model. The reason is that a loosely-timed simulation takes place in the system-level profiling procedure, and then, the static analysis is executed repeatedly to estimate the performance of architectures. Therefore, one of the most obvious advantages of the proposed method over the dynamic simulation methods, e.g., RTL, CA and BCA simulation, is that it requires less modeling effort. Unlike dynamic simulations, in which models for each architecture must be implemented, the SLM in the proposed method is created and profiled only once and the information can be utilized for performance estimation of every ALM. Consequently, days of modeling effort can be saved because inferior architectures are discriminated in an early design stage.

Figure 14 illustrates the logarithm relationship between the number of AL-EDG vertices, n, and the runtime of the proposed method spent on AL-EDG analysis for eight individual architectures encoding the 512×512 pixel-image. The slopes of lines in the graph shows that the runtime of the experiments increases by only $O(n^{1.65})$. The reason is that in the conducted estimations, the number of executable vertices in each executable vertex set of the functional blocks, DMA controllers, memories, shared buses and buses on the bus matrix of the multi-layer bus in an architecture is scheduled to as few as no more than five process or channel vertices during each iteration. Consequently, the complexity of the most complex step, ordering the executable vertices, is reduced to n and so does the overall complexity which becomes n^2.

Considering the speed-up value, the proposed method is fast and uses less memory resource comparing to the conventional RTL simulation. The efficient performance estimation method is able to evaluate the performance of the architecture encoding a larger image in a short time when the RTL simulation takes too much time. Moreover, RTL simulation might be impossible because the memory resource is insufficient in some circumstances, while the proposed method can overcome such problem due to the fact that the method refrains from using the real image data.

In Sections 5.3 and 5.4, experiments were conducted only to the architectures with the same set of functional blocks, DMA controllers, and memories. The proposed estimation method achieves 25.6 times faster estimation compared to RTL simulation with small error. Furthermore, proposed method also works well with different sets of components by repeating only the AL-EDG related procedures.

The runtime of the proposed method is roughly compared with the CA simulation time in order to approximate the speed-up. The experiments were conducted again on the Pentium4 workstation, running at 3.4 GHz. Then, the results are evaluate against the CA simulation time results presented by Martin et al. [17]. It is found that the proposed method has gained approximately 30-35 times of runtime speed-up over the CA simulation.

6. Conclusion

This paper proposes an efficient performance estimation method for a configurable multi-layer bus-based architecture by utilizing system-level data flow information. The flow of performance analysis takes the outstanding behavior details of bus protocol into account so that it can recognize the dynamic bus contention. The proposed method is fast and accurate. It estimates the performance within 8% of error comparing to the conventional RTL simulation. Furthermore, the AL-EDG construction and analysis have achieved the speed-up of 152.6 times over RTL simulation in estimating the performance of one architecture. The experimental results also shows that the proposed performance estimation method including system-level profiling, SL-EDG construction, and AL-EDG construction and analysis of eight architectures has achieved the overall speed-up of 25.6 times over the eight RTL simulations. When evaluating more architectures, the proposed method repeats only the AL-EDG construction and analysis procedures. Therefore, the larger the number of architectures becomes, the bigger overall speed-up value is gained.

In the future, the statistical analysis shall be applied to the proposed method to assure the estimated performance statistically and propose a system-level architecture optimization framework to explore multi-layer bus-based architectural platform.

Acknowledgments This research was partly supported by the Ministry of Education, Culture, Sports, Science and Technology, Grant-in-Aid for Scientific Research C 25330059.

References

[1] Ueda, K., Sakanushi, K., Takeuchi, Y. and Imai, M.: Architecture-level performance estimation method based on system-level profiling, *IEE P-Comput. Dig. T.*, Vol.152, No.1, pp.12–19 (2005).

[2] Verkest, D., Rompaey, K., Bolsens, I. and Man, H.: CoWare–A design environment for heterogeneous hardware/software systems, *Des. Autom. Embed. Syst.*, Vol.1, No.4, pp.357–386 (1996).

[3] Honda, S., Wakabayashi, T., Tomiyama, H. and Takada, H.: RTOS-Centric Hardware/Software Cosimulator for Embedded System Design, *Proc. CODES+ISSS '04*, pp.158–163 (2004).

[4] Balarin, F., Lavagno, L., Passerone, C., Sangiovanni-Vincentelli, A., Watanabe, Y. and Yang, G.: Concurrent Execution Semantics and Sequential Simulation Algorithms for the Metropolis Meta-model, *Proc. CODES '02*, pp.13–18 (2002).

[5] Buck, J., Ha, S., Lee, E.A. and Messerschmitt, D.G.: Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems, *Int. Journal of Computer Simulation*, Vol.4, (1994).

[6] Yoo, S., Nicolescu, G., Gauthier, L. and Jerraya, A.: Automatic Generation of Fast Timed Simulation Models for Operating Systems in SoC Design, *Proc. DATE '02*, pp.620–627 (2002).

[7] Bouchhima, A., Yoo, S. and Jerraya, A.: Fast and accurate timed execution of high level embedded software using HW/SW interface simulation model, *Proc. ASP-DAC 2004*, pp.469–474 (2004).

[8] Yoo, S. and Jerraya, A.: Hardware/software cosimulation from interface perspective, *Computers and Digital Techniques, IEE Proceedings*, Vol.152, No.3, pp.369–379 (2005).

[9] Takahashi, M., Miyajima, H. and Fukui, M.: An Efficient Power and Performance Evaluation Method with Reconfigurable Bus Architecture Model, *Proc. SASIMI 2003*, pp.345–350 (2003).

[10] Lahiri, K., Raghunathan, A. and Dey, S.: System-level Performance Analysis for Designing On-chip Communication Architectures, *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, Vol.20, No.6, pp.768–783 (2006).

[11] Loghi, M., Angiolini, F., Bertozzi, D., Benini, L. and Zafalon, R.: Analyzing On-Chip Communication in a MPSoC Environment, *Proc. DATE '04*, Vol.2, pp.752–757 (2004).

[12] Pasricha, S. and Dutt, N.: *On-Chip Communication Architectures: System on Chip Interconnect*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2008).

[13] Pasricha, S., Dutt, N. and Ben-Romdhane, M.: Fast exploration of bus-based communication architectures at the CCATB abstraction, *ACM Trans. Embed. Comput. Syst.*, Vol.7, No.2, pp.22:1–22:32 (2008).

[14] Pasricha, S., Dutt, N. and Ben-Romdhane, M.: BMSYN: Bus Matrix Communication Architecture Synthesis for MPSoC, *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, Vol.26, No.8, pp.1454–1464 (2007).

[15] Schirner, G. and Domer, R.: Quantitative Analysis of Transaction Level Models for the AMBA Bus, *Proc. DATE '06*, pp.1–6 (2006).

[16] Baganne, A., Bennour, I., Elmarzougui, M., Gaiech, R. and Martin, E.: A multi-level design flow for incorporating IP cores: Case study of 1D wavelet IP integration, *Proc. DATE '03*, pp.250–255 suppl. (2003).

[17] Martin, G.: Overview of the MPSoC design challenge, *Proc. DAC '06*, pp.274–279 (2006).

[18] Madl, G., Pasricha, S., Bathen, L.A.D., Dutt, N. and Zhu, Q.: Formal Performance Evaluation of AMBA-based System-on-chip Designs, *Proc. EMSOFT '06*, pp.311–320 (2006).

[19] Lee, C., Kim, S. and Ha, S.: A Systematic Design Space Exploration of MPSoC Based on Synchronous Data Flow Specification, *J. Signal Process. Syst.*, Vol.58, No.2, pp.193–213 (2010).

[20] Li, M., Achteren, T., Brockmeyer, E. and Catthoor, F.: Statistical Performance Analysis and Estimation for Parallel Multimedia Processing, *J. Signal Process. Syst.*, Vol.58, No.2, pp.105–116 (2010).

[21] Liu, H.-Y., Petracca, M. and Carloni, L.P.: Compositional system-level design exploration with planning of high-level synthesis, *Proc. DATE '12*, pp.641–646 (2012).

[22] Cho, Y.-S., Choi, E.-J. and Cho, K.-R.: Modeling and Analysis of the System Bus Latency on the SoC Platform, *Proc. SLIP '06*, pp.67–74 (2006).

[23] Accellera Systems Initiative: IEEE Standard for Standard SystemC® Language Reference Manual, available from ⟨http://standards.ieee.org⟩ (2012).

[24] ARM: ARM Multi-layer AHB Technical Overview (rev2.0), available from ⟨http://infocenter.arm.com⟩ (2001).

[25] ARM: AMBA Specification (Rev 2.0), available from ⟨http://infocenter.arm.com⟩ (1999).

Salita Sombatsiri received her B.E. from Chulalongkorn University in 2010 and Master of Information Science and Technology from Osaka University in 2013. She is currently a Ph.D. candidate at Osaka University. Her research interests include system level design methodology of embedded systems and the modeling of communication architecture.

Yoshinori Takeuchi received his B.E., M.E. and Dr. Eng. degrees from Tokyo Institute of Technology in 1987, 1989 and 1992, respectively. From 1992 through 1996, he was a research associate of the Department of Engineering, Tokyo University of Agriculture and Technology. From 1996, he has been with Osaka University. He was a visiting scholar in University of California, Irvine from 2006 to 2007. He is currently an associate professor of Graduate School of Information Science and Technology at Osaka University. His research interests include system level design, VLSI design and VLSI CAD. He is a member of IEICE of Japan, IPSJ, ACM, and SP, CAS and SSC Society of IEEE.

Masaharu Imai received his B.S. degree in Electrical Engineering in 1974 and M.S. and Ph.D. degrees in Information Science from Nagoya University in 1976 and 1979. From April 1979 to March 1996, he was with the Department of Information and Computer Sciences, Toyohashi University of Technology, where his final title was professor. He was a visiting professor at the University of South Carolina, Columbia, SC, from 1984 to 1985. Since April 1996, he has been with Osaka University, where he is a professor of the Department of Information Systems Engineering, the Graduate School of Information Science and Technology. His research interests include ASIP design automation, hardware/software codesign, VLSI architecture, and system level design methodology of embedded systems, Since 1991, he has been working for EDA standardization including VHDL under IEEE and Japan Electronics and Information Technology Industries Association (JEITA). He is a member of IEICE, ACM, and IPSJ.

(Recommended by Associate Editor: *Tohru Ishihara*)