Design Space Exploration of System-on-Chip Bus Architectures

2005 年 8 月

서울대학교 大學院
電氣컴퓨터 工學部
金 聖 讚
Abstract

On-chip communication is increasingly being regarded as one of the major hurdles for complex SoC designs. In particular, at the system-level, the integration of an increasing number and variety of complex components is resulting in rapid growth in the volume and diversity of on-chip communication traffic, imposing stringent requirements on the underlying on-chip communication resources. As a result, on-chip communication has started to play a significant role in determining several system-wide metrics, including overall system performance, power consumption, and reliability. In this thesis, we propose a methodology for customizing the design of the communication architecture to exploit the characteristics of on-chip communication traffics generated by application, in particular, being concerned about bus-based architectures since they are still most widely used due to its simplicity and popularity.

Unfortunately, the design space of bus-based communication architectures to consider is extremely wide since it can be formed by multiple axes such as bus topology, processing component allocation, bus arbitration, operation clock frequency, data width, and so on. Therefore, it is critical to develop an efficient exploration methodology, which is the main theme of this thesis. Fast and accurate performance evaluation method is the key to achieve this goal. In general, since
speed and accuracy are two conflicting goals of performance estimation, approaches on performance estimation had been proposed to satisfy one of them. To put it more concretely, there are two representatives satisfying one of the goals respectively, which are static performance estimation and simulation. The static performance evaluation usually relies on the static characteristics of system such as the amount of data transfer, fixed bit-width of channel for transfer and so on. The estimation based on this approach can be made quite fast, but may be inaccurate for not being able to consider dynamic behavior of communication architectures. On the other hand, simulation can yield an accurate performance estimate by executing actual system with pre-determined input stimuli. Too long run time, however, is its main drawback. To our best knowledge, no approach that satisfies both goals at the same time had been proposed until now.

In the proposed methodology, we utilize the advantages of both approaches by breaking down the exploration procedure into two steps. In the first step, we use a static performance estimation technique to quickly evaluate each candidate design point and drastically prune them in the large design space. To cope with such huge design space, we consider various kinds of parameters of on-chip bus architectures, which mainly consist of bus topology, bus protocol, and memory allocation. In order to perform this step efficiently, a queuing model for fixed priority base bus system is used. It enables an accurate performance estimation of target architecture by considering dynamic bus conflicts due to simultaneous accesses by multiple processing elements on a bus. In addition, extensions of the proposed technique are
presented to take into account not only other bus arbitration schemes, but also multitask environments. Experimental results show that the proposed estimation technique is accurate by less than 10% error compared with a simulation approach while being tens times faster.

The second step uses trace-driven simulation to accurately evaluate the design points in the reduced space and determine a pareto-optimal set of on-chip bus architectures. The proposed methodology was successfully applied to preliminary examples including 4-Channel digital video recorder and an equalizer subsystem for OFDM DVB-T receiver. The performance improvement for both examples amount to 100% in average, which implies that a significant performance gain of system can be obtained by optimizing communication architecture only. This also supports the efficiency and the viability of the proposed methodology to explore the wide design space of on-chip bus architectures.

Keywords: systems-on-chip, on-chip bus design, design space exploration, performance estimation, queuing theory, architecture optimization

Student number: 2000-30463
Contents

Abstract .......................................................................................................................... i

Contents ...................................................................................................................... iv

List of Figures ............................................................................................................ viii

List of Tables ........................................................................................................... xiii

List of Tables ........................................................................................................... xiii

List of Abbreviations ................................................................................................. xv

Chapter 1  Introduction .............................................................................................. 1

1.1  Research Challenges in Design of On-Chip Bus Architectures ............ 1

1.2  Overview of the Proposed Exploration Methodology .................... 3

1.3  Contributions ........................................................................................................ 8

1.4  Thesis Organization ........................................................................................... 10

Chapter 2  Background of On-Chip Communication Architectures ........ 11

2.1  Terminology ......................................................................................................... 12

2.2  Communication Architecture Topologies ......................................................... 13

2.3  On-Chip Communication Protocols ................................................................. 16

2.4  Communication Interfaces ................................................................................ 18
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3.1</td>
<td>Introduction</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>3.2</td>
<td>Related Work</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>3.3</td>
<td>Queuing Model of Fixed Priority based Bus</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>3.3.1</td>
<td>Queuing Model of Single I/O Bus</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>3.3.2</td>
<td>Extension to Processor Bus</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>3.4</td>
<td>Model of Round-Robin based Bus</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>3.5</td>
<td>Model of Two-Level TDMA based Bus</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>3.6</td>
<td>Chapter Summary</td>
<td>49</td>
</tr>
<tr>
<td>4</td>
<td>4.1</td>
<td>Estimation of Single Bus</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>4.2</td>
<td>Extension to Multiple Bus Systems</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td>4.3</td>
<td>Time Complexity</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>4.3.1</td>
<td>Comparison with Trace-Driven Simulation</td>
<td>61</td>
</tr>
<tr>
<td></td>
<td>4.3.2</td>
<td>Effects of Schedule Complexity</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>4.4</td>
<td>Experiments</td>
<td>66</td>
</tr>
<tr>
<td></td>
<td>4.4.1</td>
<td>Estimation Accuracy for Single Bus Systems</td>
<td>66</td>
</tr>
<tr>
<td></td>
<td>4.4.2</td>
<td>Comparison with Simpler Models</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td>4.4.3</td>
<td>Evaluation of 4-Channel DVR</td>
<td>73</td>
</tr>
</tbody>
</table>
Chapter 7   Conclusions and Future Directions ........................................ 121

Bibliography ..................................................................................................... 124
List of Figures

Figure 1-1. (a) The behavior specification of an illustrative example, (b) its single shared bus implementation, and (c) and (d) dual-bus implementations with two different mappings of shared memory segments $SM_{arc2}$ and $SM_{arc3}$.....................4

Figure 1-2. The proposed design space exploration flow...........................................5

Figure 3-1. The performance variation due to dynamic bus conflicts on a single bus for two bus request rates: (a) 0.115 (typical for multimedia applications) and (b) 0.17 (highly intensive cases)...................................................................................22

Figure 3-2. The queuing model of a single bus.......................................................27

Figure 3-3. The state transitions from the state $(0,1,1,h)$ by an occurrence of single event: new arrival of request from each priority class or completion of a request on a server....................................................................................................................31

Figure 3-4. The pseudo code describing the iterative procedure of the base queuing model. ................................................................................................................................................33

Figure 3-5. The average estimation error of the compromised queuing model compared to the general queuing models with the various numbers of simultaneous events. ..................................................................................................................................39
Figure 3-6. The variations of estimation error of the compromised queuing model compared to the general queuing models with the various numbers of simultaneous events. ......................................................................................................................39

Figure 3-7. The execution time to solve the compromised queuing model and the general queuing models with the various numbers of simultaneous events. ..........40

Figure 3-8. Two alternatives of round-robin arbitration......................................42

Figure 3-9. The state transition diagram of a round-robin bus. .........................43

Figure 3-10. (a) An example of TDMA slot assignment and (b) the state transitions from $bs_{rT}(0, I)$ according to the slot assignment in (a).................................................47

Figure 4-1. (a) An example schedule of $PE_0$, $PE_1$, and $PE_2$ and its corresponding queuing model, (b) new queuing model after function block $C$ is finished at $T_1$, and (c) another queuing model after function block $B$ is finished at $T_2$ .......................51

Figure 4-2. The schedule-aware performance estimation algorithm for a single bus. ........................................................................................................................53

Figure 4-3. The modeling of communication via a bus bridge.........................56

Figure 4-4. The schedule-aware performance estimation algorithm for multiple-bus architecture..................................................................................................................59

Figure 4-5. The comparison of time complexity between the proposed technique and trace-driven simulation using typical memory access rate: 0.12......................63
Figure 4-6. The comparison of time complexity between the proposed technique and trace-driven simulation using highly intensive memory access rate: 0.17.......63

Figure 4-7. The specification of H.263 Decoder and the mapping of function blocks to (a) an ARM9 processor and (b) a dedicated IDCT hardware. The groups in (a) and (b) are executed 99 times during decoding one frame. ........................................65

Figure 4-8. Grouping the function blocks in the schedule is required to enhance the efficiency of the proposed technique. .................................................................65

Figure 4-9. The burst transfer of four words on our bus model.........................67

Figure 4-10. (a) The schedule of function blocks on N processing elements \((PE_0, PE_1, \ldots, PE_{N-1})\) and (b) a single bus architecture that consists of N processing elements and a memory subsystem.................................................................68

Figure 4-11. The estimation accuracy according to the number of processing elements and the memory access time for each arbitration scheme.................69

Figure 4-12. The comparison of estimation errors for the example systems of Figure 4-10 between (a) the base queuing model for an I/O bus, (b) the intuitive equation, and (c) the proposed model for the processor bus.........................72

Figure 4-13. (a) The specification of an H.263 encoding algorithm, (b) the initial schedule of 4-Channel DVR. ..................................................................................74

Figure 4-14. (a) Memory traces and (b) AHB transaction by memory accesses from the time 320 to the time 323 in (a): a four-beat incrementing/wrapping burst transfer. .................................................................................................77
Figure 4-15. Estimation errors of the proposed technique compared with the AHB bus using the randomly generated memory traces. ............................................................ 78

Figure 5-1. (a) The specification and (b) the schedule of H.263 encoder and (c) the mapping of function blocks to processing elements for 4-Channel DVR. ............. 83

Figure 5-2. The function blocks of $ch1$, $ch2$, and $ch3$ on $ARM0$, $ARM1$, and $HW_DCT$, which can be executed simultaneously with $ME0$. ......................... 84

Figure 5-3. The estimated bus access time compared with those by simulation for each processing element in 4-Channle DVR. ............................................................... 88

Figure 6-1. (a) The initial schedule of 4-Channel DVR and (b) its single-bus implementation. ........................................................................................................ 94

Figure 6-2. Creating a new bus for processing element $ARM0$ and allocating its associated shared memory segments............................................................. 96

Figure 6-3. If the local memory segment of processor $ARM0$ is separated from shared bus $Bus1$, communication traffics incurred by local memory accesses can be removed from $Bus1$, which may lead to further performance improvement. ........ 96

Figure 6-4. The overall structure of trace-driven simulator.................................. 101

Figure 6-5. The pseudo code describing the trace-driven simulation. ............... 105

Figure 6-6. The proposed exploration flow. ....................................................... 108

Figure 6-7. (a) The specification of the equalizer subsystem for OFDM DVB-T receiver and (b) its schedule. .............................................................................. 109
Figure 6-8. Three example systems: (a) System1, (b) System2, and (c) System3.

Figure 6-9. Performance variations of 5 example systems during the exploration according to the number of buses. In each graph, horizontal and vertical axes represent the number of buses used and the normalized execution time respectively.

Figure 6-10. The final bus architectures of the equalizer subsystem for OFDM DVB-T receiver: (a) without considering local bus and (b) with considering local bus.

Figure 6-11. The final bus architectures of equalizer for 4-Channel DVR: (a) without considering local bus and (b) with considering local bus.
List of Tables

Table 3-1. Type 1 transitions from the state \((n_1, n_h, n_0, s)\) to other states...............36
Table 3-2. Type 2 transitions from the state \((n_1, n_h, n_0, s)\) to other states...............36
Table 4-1. The complexity of the proposed estimation technique.........................60
Table 4-2. Average estimation errors according to the estimation techniques. ......73
Table 4-3. The size of memory traces obtained from 4-Channel DVR. ...............74
Table 4-4. The experimental results about 4-Channel DVR for fixed priority......75
Table 4-5. The experimental results about 4-Channel DVR for round-robin.......76
Table 4-6. The experimental results about 4-Channel DVR for two-level TDMA. 76
Table 4-7. Estimation errors of the proposed technique compared with the AHB
bus applied to 4-Channel DVR. .................................................................78
Table 5-1. The results of exploration for 4-Channel DVR. .............................87
Table 6-1. The number of generated architectures from single-bus architecture of 4-
Channel DVR by changing allocation of shared memory segments...............97
Table 6-2. Efficiency of the priority assignment heuristic compared with an
exhaustive method: The equalizer subsystem for OFDM DVB-T receiver........111
Table 6-3. Efficiency of the priority assignment heuristic compared with an
exhaustive method: 4-Channel DVR. .............................................................111
Table 6-4. The experimental results of exploration for 5 example systems. ........114

Table 6-5. Performance improvement obtained by considering local bus exploration. ..................................................................................................................................................116
# List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AHB</td>
<td>Advanced High-performance Bus</td>
</tr>
<tr>
<td>AMBA</td>
<td>Advanced Microcontroller Bus Architecture</td>
</tr>
<tr>
<td>ASIC</td>
<td>Application Specific Integrated Circuit</td>
</tr>
<tr>
<td>ATM</td>
<td>Asynchronous Transfer Mode</td>
</tr>
<tr>
<td>BFM</td>
<td>Bus Functional Model</td>
</tr>
<tr>
<td>CA</td>
<td>Collision Avoidance</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>CSMA</td>
<td>Carrier Sense Multiple Access</td>
</tr>
<tr>
<td>DCT</td>
<td>Discrete Cosine Transform</td>
</tr>
<tr>
<td>DMA</td>
<td>Direct Memory Access</td>
</tr>
<tr>
<td>DSP</td>
<td>Digital Signal Processor</td>
</tr>
<tr>
<td>DVB-T</td>
<td>Digital Video Broadcasting – Terrestrial</td>
</tr>
<tr>
<td>FIFO</td>
<td>First-In-Fist-Out</td>
</tr>
<tr>
<td>HDL</td>
<td>Hardware Description Language</td>
</tr>
<tr>
<td>I/O</td>
<td>Input / Output</td>
</tr>
<tr>
<td>IDCT</td>
<td>Inverse Discrete Cosine Transform</td>
</tr>
<tr>
<td>Acronym</td>
<td>Definition</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
</tr>
<tr>
<td>ILP</td>
<td>Integer Linear Programming</td>
</tr>
<tr>
<td>IP</td>
<td>Intellectual Property</td>
</tr>
<tr>
<td>ISS</td>
<td>Instruction Set Simulator</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network</td>
</tr>
<tr>
<td>LP</td>
<td>Linear Programming</td>
</tr>
<tr>
<td>ME</td>
<td>Motion Estimation</td>
</tr>
<tr>
<td>NoC</td>
<td>Network-on-Chip</td>
</tr>
<tr>
<td>OFDM</td>
<td>Orthogonal Frequency Division Multiplexing</td>
</tr>
<tr>
<td>QCIF</td>
<td>Quarter Common Intermediate Format</td>
</tr>
<tr>
<td>RTL</td>
<td>Register Transfer Level</td>
</tr>
<tr>
<td>SCSI</td>
<td>Small Computer System Interface</td>
</tr>
<tr>
<td>SRAM</td>
<td>Static Random Access Memory</td>
</tr>
<tr>
<td>SoC</td>
<td>System-on-Chip</td>
</tr>
<tr>
<td>TDMA</td>
<td>Time Division Multiple Access</td>
</tr>
<tr>
<td>VHDL</td>
<td>Very-high-speed-integrated-circuit Hardware Description Language</td>
</tr>
<tr>
<td>VSIA</td>
<td>Virtual Socket Interface Alliance</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Research Challenges in Design of On-Chip Bus Architectures

On-chip communication is increasingly being regarded as one of the major hurdles for complex SoC designs [34]. As technology scales into the nanometer era, chip-level wiring presents numerous challenges, including large signal propagation delays, high-power dissipation, and increased susceptibility to errors [35][36]. On the other hand, at the system-level, the integration of an increasing number and variety of complex components is resulting in rapid growth in the volume and diversity of on-chip communication traffic, imposing stringent requirements on the
Chapter 1 Introduction

underlying on-chip communication resources. As a result, on-chip communication has started to play a significant role in determining several system-wide metrics, including overall system performance, power consumption, and reliability.

The growing importance of on-chip communication adds a new facet to the process of system design. Under the traditional notion of system design, a system’s functional requirements are refined and mapped onto a well-optimized set of computational resources and storage elements. Today, system designers are required to pay increasing attention to the relatively less understood process of mapping a system’s communication requirements onto well-optimized on-chip communication architecture. We have reached a point at which system design practices need to evolve from being computation-centric to begin increasingly communication-aware. Recognizing this, new architectures and system design techniques have started to emerge to address on-chip communication.

The problem of communication architecture design for SoCs is addressed in terms of the following steps: (1) design or selection of an appropriate network topology, which defines the physical structure of the communication architectures; (2) design of the on-chip communication protocols; and (3) communication mapping, which specifies a mapping of system communications (dataflow) to physical paths in the communication architecture.

This thesis describes advances in a design methodology for communication architecture design that is concerned with the above problems. We point out the significant advantages of performance that can be obtained by customizing the
communication architecture to characteristics of the communication traffic generated by the SoC components. An important requirement for any communication architecture design and customization environment is the availability of automatic techniques for analyzing the impact of the communication architecture (including the effects of the topology, protocols, and communication mapping) on system performance. We propose the techniques for customizing the design of the communication architecture to exploit the characteristics of the on-chip communication traffic generated by application, in particular, being concerned about bus-based communication architectures since it is still most widely used due to its simplicity and popularity.

1.2 Overview of the Proposed Exploration Methodology

For better understanding of the proposed methodology, we use an illustrative example of Figure 1-1. Initially, the system behavior is specified as a block diagram of four functions blocks. The arcs between function blocks show the data dependency. For example, function blocks B and D can be executed only after function block A is completed. Those four function blocks are mapped to three processing elements: A and C are mapped to processing element PE0, B to PE1, and D to PE2 respectively.
Figure 1-1. (a) The behavior specification of an illustrative example, (b) its single shared bus implementation, and (c) and (d) dual-bus implementations with two different mappings of shared memory segments $SM_{arc2}$ and $SM_{arc3}$.

Figure 1-1(b) represents a single shared bus implementation. Note that one physical memory component is connected to a bus and it contains seven logical memory segments: Three local memory segments and four shared memory segments. Memory segments $LM_{PE0}$, $LM_{PE1}$, and $LM_{PE2}$ are local memory segments of PE0, PE1, and PE2 respectively. The arcs between function blocks are implemented as shared memory segments for inter-component communication. For example, $SM_{arc0}$ associated with arc0 in Figure 1-1(a) indicates a shared memory segment for communication between function blocks A and B.
Chapter 1  Introduction

The proposed exploration technique, whose overall structure is shown in Figure 1-2, starts the exploration process with this single-bus architecture that becomes the only element in the “set of architecture candidates” initially. A set of memory traces is one of the inputs to the proposed exploration procedure, which includes both local and shared memory accesses from all processing elements. After the mapping of function blocks to processing elements is completed, the memory traces are obtained using instruction set simulator for each processor core, HDL simulator for ASIC parts, or IP simulators. Memory traces are classified into 3 categories: Code memory, data memory and shared memory. Code and data memories are associated with local memory accesses and shared memory with
inter-component communication. For the processor that uses a cache memory, the traces associated with code and data memories represent the memory accesses incurred by cache-miss.

We traverse the design space in an iterative fashion as shown in Figure 1-2. The body of the iteration loop consists of three main steps. The purpose of the first step is to quickly explore the design subspace of architecture candidates to build a reduced set of design points to be carefully examined. With a given set of architecture candidates, we visit all design points by varying the priority assignment of processing elements on each bus and other bus operation conditions.

To perform this step more effectively, we developed a static performance evaluation technique based on a queuing theory. Since a bus is a shared medium among multiple processing elements that compete each other for using it, communication overhead is highly unpredictable due to bus contention. As more processing elements are integrated into a single SoC and application become more complicated to require significant communication traffics, it is expected that the impact of dynamic conflict on entire system performance would increase rapidly. In other words, the overhead due to bus conflicts becomes the dominating factor on system performance as the number of processing elements increases and the bus request rate becomes higher. It is, thus, critical to consider the bus conflicts for accurate performance estimation. In this regard, we propose a technique based on the queuing model of target bus architecture where processing elements are customers and a bus with associated memories is a single server. Since this model
Chapter 1  Introduction

aims at accurate estimation of dynamic conflict due to simultaneous accesses of multiple processing elements on a bus, it gives us reasonably accurate estimation results to be used as the first-cut pruning of the design space. Consequently, using static estimation, we collect the design points with performance differences by less than 10% compared with the best because the proposed static estimation method has less than 10% error bound as reported in Section 4.4.

The second step applies trace-driven simulation to the selected design points from the first step. It accurately evaluates the performance of design points in the reduced space and determines the best design point. If the performance of the best design point is not improved from the previous iteration, we exit the exploration loop. Otherwise, we go to the third step and repeat another round of iteration.

The third step generates the next set of architecture candidates: From the architecture of the best design point, we explore the design space incrementally by selecting a processing element and allocating it to a different bus or a new one. Let us go back to the example of Figure 1-1. Since the first round starts with one architecture candidate, single-bus architecture, it becomes the input architecture to the third step. Suppose that PE2 is selected and allocated to a new bus to make a dual-bus system. Since all local memory segments should reside in the same bus as the associated processing element, there are 4 candidate architectures depending on where to put the shared memory segments associated with PE2: SM_arc2 and SM_arc3. Function blocks A and D use shared memory segment SM_arc2 so that it may be allocated either to Bus0 or Bus1. However SM_arc0 that is accessed by
Chapter 1  Introduction

function blocks A and B should remain at Bus0 since all of its associated processing elements reside in the same bus. Among four candidate architectures, Figure 1-1(c) and Figure 1-1(d) show two candidate architectures. In the case we select PE0 and move it to a new bus, we generate 16 different candidate architectures since PE0 is associated with four shared memory segments. In this way, we can generate 24 candidate architectures for the second round of iteration by moving a processing element into a new bus and considering all possible shared memory segment allocation.

As the iteration goes, we record the best performance numbers as a function of the number of buses to obtain the pareto-optimal design points. If the number of buses increases, the performance tends to increase. We exit the iteration when no performance increase is obtained from the previous iteration.

The proposed technique does not explore the entire design space but it is a greedy heuristic to prune the design space aggressively since we select only the best architecture at the end of iteration. If we select multiple ones, we may explore the wider set of design points with longer execution time.

1.3 Contributions

The contributions of this thesis can be summarized as follows:
Chapter 1   Introduction

- We propose the efficient static estimation technique using a queuing theory to take into account dynamic behavior due to bus contentions. The queuing model is also extended to model bus architectures with various kinds of arbitration scheme as well as popular static priority based arbitration.

- By adopting the static schedule of application, we make the accuracy of the estimated entire execution time comparable with trace-driven simulation. This also enables designers to explore the wide design space of communication architectures considering the more design axes than previous works.

- We explore the larger design space considering multiple design axes such as the number of buses, bus topology, component allocation, priority assignment and other bus operating conditions. Since the proposed exploration technique is systematic and extensible, more design axes can easily be added. In addition, we consider local memory accesses as well as shared memory accesses. Previous works on architecture exploration had been mostly concerned with only communication requirements between processing elements ignoring local memory accesses. Since local memory accesses are also involved in bus contention, they need to be considered.

- The techniques described above are integrated into a single framework to systematically automate the proposed design flow. It was successfully applied to various applications including 4-Channel digital video recorder and the equalizer subsystem for OFDM DVB-T receiver. Through those examples, we were able to show that a significant performance gain of system can be obtained by optimizing
communication architecture only. In conclusion, we show the importance of on-chip bus architecture for successful SoC design and provide the efficient methodology to solve it.

1.4 Thesis Organization

The rest of the thesis is organized as follows. In Chapter 2, preliminaries on on-chip communication architectures are given, which include architecture topologies, bus protocols, and communication interface synthesis. Chapter 3 discusses a queuing model of bus architecture considering dynamic conflicts for various kinds of arbitration schemes and its extension to multiple bus systems. In Chapter 4, a performance estimation technique taking into account the queuing model and static schedule of given application is explained. Subsequently, some quantitative analysis on time complexity is provided. Chapter 5 contains an extension of the performance estimation technique to multitask applications. An exploration framework employing all techniques in the previous three chapters is introduced in Chapter 6. Finally, we draw conclusions and address future directions in Chapter 7.
Chapter 2

Background of On-Chip Communication Architectures

In this chapter, for convenience of the readers, we provide some background of on-chip communication architectures, the original literature of which can be found in [37]. The on-chip communication architecture refers to a fabric that integrates SoC components and provides a mechanism for the exchange of data and control information between them. The first basis for classifying communication architectures is the network topology, which defines the physical structure of the communication architecture. In general, the topology could range in complexity from a single shared bus to an arbitrary complex, regular or irregular interconnection of channels. The second basis for classification is the
communication protocols employed by the communication architectures, which specify the exact convections and logical mechanism according to which system component communicate over the communication architecture. In addition, these protocols define resource management and arbitration mechanisms for accesses to shared portions of the communication architecture. As mentioned in the previous chapter, communication mapping refers to the process of mapping the system-level communication events to physical paths in the topology. In the rest of this section, we survey existing and emerging communication architecture topologies and communication protocols. We also describe recent advances in communication interface design.

2.1 Terminology

Two kinds of components may be connected to the communication architecture. Masters are system components that can initiate communications (reads/writes). Examples include CPUs, DSPs, DMA controllers, and so on. Slaves are system components (e.g. on-chip memories and passive peripherals) that do not initiate communication transactions by themselves but merely respond to transactions initiated by a master. Bridges or routers allow communication between component pairs that are connected to different communication Channels. Bridges may support one-way or two-way communications and may themselves consist of multiple master and slave interfaces. Communication resource management and arbitration
algorithms are implemented in centralized or distributed arbiters or controllers. Numerous parameters help define properties of the communication Channels and their associated protocols, such as bus widths, burst transfer size, and master priorities.

### 2.2 Communication Architecture

#### Topologies

In this section, we describe existing and emerging communication architecture topologies, in order of increasing sophistication and complexity.

**Shard bus**

The system bus is the simplest example of a shared communication architecture topology and is commonly found in many commercial SoCs (e.g. PI-bus [38]). The bus consists of a set of address, data, and control lines shared by a set of masters, that contend among themselves for access to one or more slaves. In this simplest form, a bus arbiter periodically examines accumulated requests form the multiple master interfaces, and grants access to a master using arbitration mechanisms specified by the bus protocol. However, limitations on bus bandwidth (due to
increasing load on global bus lines) are focusing system designers to use more advanced topologies.

**Hierarchical Bus**

In recognition of the problems associated with topologies based on a flat-shared bus, topologies consisting of multiple busses have recently begun to appear. This architecture usually consists of several shared busses interconnected by bridges to form a hierarchy. Different levels of the hierarchy correspond to the varying communication bandwidth requirements of SoC components. Commercial examples of such architecture include the AMBA bus architecture (ARM) [3] and the CoreConnect Architecture (IBM) [24]. Different levels of the hierarchy are connected by bridges. Transactions across the bridge involves additional overhead, and, during the transfer, both busses remain inaccessible to other components. However, multiple word communication can proceed across the bridge in a pipelined manner. Such topologies typically offer large throughput improvements over the shared topology, due to decreased load per bus, and the potential for transactions to proceed in parallel on different busses.

**Rings**

In certain application areas, ring-based communication architectures have been used, such as high-speed ATM switches [39]. In such architectures, each
component (master/slave) communicates using a ring interface, which typically implements a token-passing protocol. The advantage of ring-based architectures is that each communication channel is of shorter length (point to point between neighboring component) and therefore can potentially support higher clock speeds.

**Packet Switched Fabrics**

Recently, the use of more complex topologies has been proposed for SoCs consisting of large numbers of processing elements. These architectures avoid the use of globally shared buses but rely on switching mechanisms to multiplex communication resources among different master and slave pairs. Although these architecture have been studied in detail in the context of general-purpose systems such as multiprocessors and network routers, examples of such emerging architectures in the SoC domain include architectures based on crossbars [39], fat-trees [40], octagons [41], and two-dimensional meshes [42][43]. The advantages such architectures are expected to provide include higher on-chip communication bandwidth and predictable electrical properties, due to regularity in the communication architecture topology.
2.3 On-Chip Communication Protocols

In this section, we describe existing and emerging communication protocols, focusing on the different types of resource management algorithm employed for determining access rights to shared communication Channels. In this regard, these on-chip communication protocols are analogous to those used in the “datalink” layer of wide area networks, which determine access right to shared LAN resources (e.g. CSMA/CA) [44]. As systems continue to grow in complexity, it is expected that other, higher layer protocol concepts from the domain of large scale networking will become increasingly applicable to communication architectures, such as routing, flow control, and quality of service [45].

Static Priority

This is a commonly employed arbitration scheme technique used for shared bus-based communication architectures [3][24][38]. In this protocol, a centralized arbiter examines accumulated requests from each master, and grants access to the requesting master that is highest priority. Transactions may be non-preemptive (i.e. once a transaction of multiple bus words begins, it runs to completions, during which time, all other components requiring access to the bus are forced to wait) or preemptive (lower priority components are forced to relinquish the bus if higher priority components are waiting). Although this protocol can be implemented
efficiently, it is not suited to providing fine-grained control over the allocation of on-chip communication bandwidth to different system components.

**TDMA**

In this type of architecture, the arbitration mechanism is based on a timing wheel with each other slot statically reserved for a unique master. Technologies are typically employed to alleviate the problem of wasted slots (inherent in time-division multiple access [TDMA]-based approaches). One approach is to support a second level of arbitration. For example, the second level can keep track of the last master interfacing to be granted via the second level and issue a grant to the next requesting master in a round-robin fashion. A commercial example of a TDMA based protocol is one offered by Sonics [21][46].

**Lottery**

In this architecture, a centralized lottery manager accumulates requests for ownership of shared communication resources from one or more masters, each of which is (statically or dynamically) assigned a number of “lottery tickets” [47]. The lottery manager probabilistically chooses one of the contending masters to be the winner of the lottery and grants access to the winner for one or more bus cycles, favoring masters that hold a larger fraction of lottery tickets. This architecture provides fine-grained control over the allocation of communication bandwidth to
system components and fast execution (low latencies) for high-priority communications, at the cost of more complex protocol hardware.

**Token Passing**

Token passing protocols have been used in ring-based architectures [39]. In such protocols, a special data word circulates on the ring, which each interface can recognize as a token. An interface that receives a token is allowed to initiate a transaction. If the interface has no pending request, it forwards the token to its neighbor. If it does have a pending request, it captures the token and reads/writes data from/to the ring, one word per cycle, for a fixed number of cycles. When the transaction completes, the interface releases the token.

### 2.4 Communication Interfaces

An important issue in designing components for use SoC designs is the use of a consistent communication interface (across different on-chip busses and communication architectures) to facilitate plug-and-play design methodologies. Using such interface can provide the additional advantages of freeing the SoC component designer from having to be aware of the details of the communication architecture to which the components will be connected. Hence, this approach facilitates the development of innovative communication architectures that are not
Chapter 2  Background of On-Chip Communication Architectures

constrained by the interfacing requirements of system components that it may potentially need to serve. According to an analogy drawn in Zhu and Malik [15], communication interface provide an abstraction that is similar to the instruction set architectures of a microprocessor, whose purpose is to make microarchitectural details transparent to the programmer and, at the same time, facilitate advances in microarchitecture design. Currently, the standards being proposed by various industry consortia in an effort to help realize this goal include interfaces based on VSIA’s Virtual Component Interface [48], and the Open Core Protocol [49].

In addition to such standardization initiatives, a large body of research has examined various issues in the design and implementation of on-chip communication interfaces, including techniques for interface modeling, simulation, and refinement (or synthesis). Thesis techniques aim at providing ways of exploring different interface, and converting high-level protocol descriptions into efficient hardware implementations. Techniques for model refinement of communication interfaces are described in [13][50], and techniques for synthesizing interface between components that feature incompatible protocols are described in [51]. Generic module interfaces (“wrappers”) are described in [52] that facilitate simulation of a complex SoC, while providing for the flexibility of mixing multiple levels of abstraction during simulation. Interface synthesis using these wrappers is described in [53].
Chapter 3  
Queuing Model of On-Chip Bus Architectures  

3.1 Introduction  

Insatiable demand of system performance makes it inevitable to integrate more and more processing elements in a single SoC to meet the performance requirements. As a new design paradigm for such high performance SoCs, the separation between function and architecture and between communication and computation is recently proposed [1][2]. Adapting this paradigm, it is assumed that
Chapter 3  Queuing Model of On-Chip Bus Architectures

system behavior is modeled as a composition of function blocks. And communication architecture is determined after a decision is made on which processing elements are used and which function blocks are mapped on which processing elements. Therefore, it allows designers to explore communication architectures independently of component selection and mapping.

While diverse interconnection networks are searched for, particularly in the realm of NoC design, we are concerned about bus-based communication architectures since it is still most widely used due to its simplicity and popularity. However, even after a specific bus standard is chosen, the design space of bus architectures can still be huge. For example, we need to determine how many bus segments are used with what topologies and which processing elements and memory banks are allocated to which bus segments. We also have to decide memory types and memory system configurations. If we include the selection of bus operation clock frequency and arbitration policy, the design space explodes.

Since a bus is a shared medium between multiple processing elements that compete each other for using it, communication overhead is highly unpredictable due to bus contention. Figure 3-1 shows the performance variation due to such dynamic conflicts in terms of bus clock cycles on the single bus complying with the AMBA AHB specification [3] varying the number of processing elements. Each processing element issues bus requests randomly with a given request rate. The request rate of 0.1 means that 10% of total execution time is used for bus accesses. The average request rate of processing elements in Figure 3-1(a) is 11.5%,
which is a typical bus request rate in multimedia applications such as H.263 encoder/decoder, MP3 decoder, and so on.

Figure 3-1. The performance variation due to dynamic bus conflicts on a single bus for two bus request rates: (a) 0.115 (typical for multimedia applications) and (b) 0.17 (highly intensive cases).
Figure 3-1(b) shows the case of the average request rate 17%, which can be thought as intensive bus traffic. The execution time of each processing element is divided into two sections: white and dark. The dark section indicates the waiting time for bus grant while the white section shows the actual execution time including the bus access time. The figure shows that the overhead due to bus conflicts becomes the dominating factor on the entire execution time as the number of processing elements increases and the bus request rate becomes higher. So it is critical to consider the bus conflicts for accurate performance estimation.

In order to explore the wide design space, we need to estimate the performance of communication architectures fast as well as accurately. While a simulation-based approach is widely used for accurate estimation, it is too slow to explore the huge design space. To overcome this drawback, we split communication architecture exploration into two steps using static performance estimation and trace-driven simulation. We focus on the static performance estimation technique, which is used to prune the design space drastically before the trace-driven simulation technique is applied. As the static estimation gets more accurate, the design space becomes smaller. The proposed static estimation technique computes the expected waiting time due to bus contention through the queuing analysis.

The remainder of this chapter is organized as follows. Section 3.2 reviews some related works. In Section 3.3, 3.4, and 3.5, the static estimation methods based on the queuing models for fixed priority, round-robin, and two-level TDMA arbitration based bus systems are explained respectively. Section 4.2 contains the
extension to multiple bus systems. Afterwards, a summary on this chapter is drawn in Section 3.6.

3.2 Related Work

Some researchers have considered communication architecture selection simultaneously during the synthesis of the computation parts of a system and the mapping step. Since the communication overhead is needed for the mapping decision, the static estimation of communication architecture has been investigated. Knudsen and Madsen estimated the communication overhead taking into account the data transfer rate variation depending on protocol, configuration, and different operating clock frequencies of components [4][5]. A technique has been proposed to estimate the communication delay using the worst-case response analysis of the real-time scheduling [6]. Ortega and Borriello took into account the static information such as data transfer size, bus protocol overhead, bus bandwidth, and so on, to estimate the worst-case bus delay [7]. Nandi and Marculescu proposed the performance measure technique based on a continuous time Markov process [8]. Daveau et al. considered only static information, such as maximum bandwidth of Channel, average and peak bandwidth of a processing element, to estimate the performance of communication links between processing elements [9]. Drinic et al. used the profiled statistics of communication traffic between cores for a given application for core-to-bus assignment [10]. Thepayasuwan and Doboli proposed
the bus architecture synthesis technique that minimizes the cost considering bus topology, communication conflict, and bus utilization using a simulated annealing [11]. However, these techniques do not model the dynamic effects such as bus contention and explore only the limited configuration space.

For exploration of communication architectures, a simulation-based estimation is widely adopted in many academic researches [12][13][14][15] and commercial tools at various abstraction levels, at the transaction level [16][17] or at the pin-level [18]. The simulation-based method gives accurate estimation results but pays too heavy computational cost to be used for exploring the large design space. So the researches based on this method cannot but exploit only a few design axes to reduce the design space.

Lahiri et al. presented the hybrid approach combining static estimation and simulation approach [19]. In their work, communication and computation segments are grouped to make a Bus and Synchronization Event (BSE) graph from the trace data obtained after system cosimulation. They focused on inter-component communication activities that are usually localized in time at the boundary of computation segment. The trace groups are scheduled on a communication media, being shifted by the estimated delays considering the resource contention. They use some static analysis to group the traces and apply a trace-driven simulation with the trace groups. Their approach is similar to ours in that they apply some static analysis to the traces to reduce the time complexity of the trace-driven simulation. However, their approach converges to the trace-driven simulation as the memory
traces become larger since the BSE graph size is dependent on the memory traces. On the other hand, our proposed technique extracts only the statistical parameters from the traces. Therefore, the run time of our technique is independent on the trace size.

Our work is inspired by Brandwajn’s work [20] where a simple queuing model of SCSI bus is proposed, which is summarized in Section 3.3.1. The model produces remarkable results compared with the simulation results. Since the communication behavior of a processor bus is quite different from an I/O bus, however, their approach cannot be directly applicable. Instead, we make several extensions to improve the estimation accuracy significantly. First, the queuing model itself is modified for the processor bus system. Second, based on the fixed priority model, we develop a novel way of modeling other types of buses such as round-robin and two-level TDMA buses. The extended models are explained in Section 3.3.2, 3.4, and 3.5 respectively. Third, we make use of the task schedule information by the aid of system level specification to consider the burstness of bus requests. Finally, the model of a single bus is extended to a multiple bus system, which makes the proposed estimation technique viable for communication architecture exploration considering various bus topologies. The details are given in Section 4.2.
3.3 Queuing Model of Fixed Priority based Bus

3.3.1 Queuing Model of Single I/O Bus

A base estimation technique using a queuing model for a fixed priority based I/O bus is reviewed in this section. The basic idea and the notations used here are borrowed from [20]. There are $N$ processing elements ($PE_0, PE_1, \ldots, PE_{N-1}$) competing for the use of a bus. It is assumed that bus arbitration is based on the fixed (or static) priorities of processing elements. $PE_0$ is assigned the highest priority. The bus access is assumed to be non-preemptive.

![Figure 3-2. The queuing model of a single bus.](image)
Figure 3-2 shows the queuing model of single bus architecture. $\lambda_i$ denotes the rate at which the processing element $PE_i$ issues memory requests. It is computed as the ratio between the memory access counts and the scheduled length of execution. If the execution time is lengthened due to bus contention, the effective arrival rate of requests becomes smaller than $\lambda_i$. We denote the actual memory access rate by $\theta_i$, which is actually seen on the bus. The mean service rate of a server for the request from $PE_i$ is denoted by $\mu_i$ and its mean service time is the reciprocal of the service rate i.e. $1/\mu_i$. Let $k_i$ be the expected number of requests from $PE_i$ waiting for use of the bus. It is within the range of $[0, 1]$ if $PE_i$ does not issue the next memory request until the current request is served. And we denote $w_i$ as the expected waiting time of the stalled request. Then, we obtain the following equation:

$$\theta_i = (1 - k_i - u_i) \cdot \lambda_i,$$  \hspace{1cm} (3-1)

where $u_i = \theta_i / \mu_i$ is the bus utilization factor of $PE_i$. Little’s law [54] says

$$k_i = w_i \cdot \theta_i \ i.e., \ w_i = \frac{k_i}{\theta_i}.$$ \hspace{1cm} (3-2)

We want to obtain $w_i$ from Equation (3-1), which indicates the delays incurred from bus contention. We can extract $\lambda_i$ from the memory traces. By the memory system and the average burst length of the memory traces, $\mu_i$ is determined statically. There remains an unknown parameter $k_i$ in the right side of Equation (3-1). To obtain this, we use a state transition diagram and its steady state probability.
Chapter 3  Queuing Model of On-Chip Bus Architectures

As the simple model of a single bus with \( N \) processing elements, each processing element has one of 3 states: computation (no bus request), waiting for bus grant, and bus access. And a bus lies in one of \( N+1 \) states depending on which processing element is currently using the bus, including the idle state. These \( N \) processing elements and the bus compose the state space of the bus system by \( N+1 \) tuples. Although this model is accurate, the number of states explodes exponentially with the increase of \( N \). For example, if \( N \) equals to 10, a system has \( 3^{10\times11} \) or 590,490 states. It means that this simple model cannot be used for fast architecture exploration. Thus we use an approximate but effective state transition diagram while preserving the accuracy within a certain limit.

We draw a state transition diagram from the viewpoint of each \( PE_i \) with \( i \in \{1, \ldots, N-2\} \). The system state is defined as a quadruple and its steady state probability by \( p(n_i,n_h,n_l,s) \), where there are \( n_i \) (\( n_i \in \{0,1\} \)) requests of \( PE_i \), \( n_h \) (\( n_h \in \{0,\ldots,i-1\} \)) requests of the processing elements with a higher priority than \( PE_i \), and \( n_l \) (\( n_l \in \{i+1,\ldots,N-1\} \)) requests of the processing elements with a lower priority than \( PE_i \) and \( s \) (\( s \in \{'i' ,'h' ,'l' ,'idle' \} \)) is the priority group that uses the bus currently. We also define a set of processing elements with a higher priority as \( \Phi_h \) and a set of processing elements with a lower priority as \( \Phi_l \) respectively. The total number of processing elements in \( \Phi_h \) and \( \Phi_l \) are defined by \( N_h \) and \( N_l \) which equal to \( i \) and \( N-i-1 \) respectively.
The state \((n_i, n_h, n_l, s)\) moves to the state \((n_i, n_h+1, n_l, s)\) if another request arrives from a processing element in \(\Phi_h\) and its transition rate is approximated by \(\alpha_h\),
\[
\alpha_h \approx (N_h - n_h) \cdot \gamma_h,
\]
where \(\gamma_h\) is the average rate of arrivals for the requests from \(\Phi_h\). \(\gamma_h\) can be computed by summing up Equation (3-1) from \(PE_0\) to \(PE_{i-1}\).
\[
\sum_{j=0}^{i-1} \theta_j = \left( N_h - \sum_{j=0}^{i-1} (k_j + u_j) \right) \cdot \gamma_h. \tag{3-3}
\]

The transition rate \(\beta_h\) to the state \((n_i, n_h-1, n_l, s)\) is approximated as follows:
\[
\beta_h \approx \sum_{j=0}^{i-1} \mu_j \cdot \frac{u_j}{\sum_{k=0}^{i-1} u_k} \cdot \gamma_h. \tag{3-4}
\]

The similar equations can be given to the requests from \(\Phi_l\); we, thus, get the followings:
\[
\sum_{j=i+1}^{N-1} \theta_j = \left( N_l - \sum_{j=i+1}^{N-1} (k_j + u_j) \right) \cdot \gamma_l \tag{3-5}
\]
and
\[
\beta_l \approx \sum_{j=i+1}^{N-1} \mu_j \cdot \frac{u_j}{\sum_{k=i+1}^{N-1} u_k} \cdot \gamma_l. \tag{3-6}
\]

Figure 3-3 shows the state transitions from the state \((0,1,1,h)\) to others. Note that not all transitions are always allowed to happen. For example, the transition from
(0,1,1,h) to (1,1,1,h) occurs whenever PE<sub>i</sub> issues a new bus request with the rate of $\lambda_i$. If current state is (1,1,1,h), such transition is not allowed since PE<sub>i</sub> has already issued a bus request, i.e. the first element of the state is ‘1’. Another transition from (0,1,2,l) to (0,1,1,h) happens when a server finish serving a request from $\Phi_l$. Therefore, the total number of requests from $\Phi_l$ decreases by 1, i.e. 2 to 1 of the third element in this transition. Subsequently, a bus is granted to $\Phi_l$, i.e. l to h in the last element of the state.

**Figure 3-3.** The state transitions from the state (0,1,1,h) by an occurrence of single event: new arrival of request from each priority class or completion of a request on a server.

From the state transition diagram and the transition rates, we can compute the steady state probability $p(n_i,n_h,n_l,s)$ using the following two conditions:
Chapter 3  Queuing Model of On-Chip Bus Architectures

- The amount of incoming transitions and that of outgoing transition are same at any state in a steady state of system.
- The total of all steady state probability for system equals to 1.

Referring the state transitions of Figure 3-3 again, we get the followings by the first condition above.

\[ \beta_i \cdot P(0,1,0,h) + \mu_i \cdot P(1,1,1,i) + \alpha_i \cdot P(0,1,2,l) + \beta_h \cdot P(0,2,1,h) = \lambda_i \cdot P(0,1,1,h) + \alpha_h \cdot P(0,1,1,h) + \alpha_i \cdot P(0,1,1,h) + \mu_i \cdot P(0,1,1,h) . \]  \hspace{1cm} (3-7)

The left side of the equality sign in Equation (3-7) is the amount of incoming transitions to the state \((0,1,1,h)\) while the right side is associated with outgoing transitions from \((0,1,1,h)\). Such equality should be constructed for all allowed states to obtain a linear system, which includes an additional equation by formulizing the second condition as follows:

\[ \sum_{n_i} \sum_{n_h} \sum_{n_l} \sum_{s} p(n_i, n_h, n_l, s) = 1 . \]

After all steady state probabilities are computed from the linear system, we can compute the expected number of waiting requests \(k_i\) by summing up the probabilities of a certain set of states as follows:

\[ \sum_{n_h} \sum_{n_l} \sum_{s} p(1, n_h, n_l, s) = u_i + k_i . \]  \hspace{1cm} (3-8)

Note that the evaluation of \(p(n_i, n_h, n_l, s)\) needs \(k_i\) by Equation (3-3) and Equation (3-8). Therefore, these equations should be solved by an iterative method until \(k_i\)
becomes stable. This iterative procedure to get \( k_i \) is repeated for each priority level \( i \ (i \in \{1, \ldots, N-2\}) \) as described in Figure 3-4. The procedure \texttt{Get\_Mean\_Waiting\_Time} takes a list of processing elements, \texttt{pe\_List}, and the statistical information, \texttt{stats\_params}, from memory traces as inputs. Then the variables \( \lambda_i \) and \( \mu_i \) are initialized using \texttt{stats\_params} in line 4. Arbitrary values within \([0, 1]\) can be used to initialize \( \theta_i \) and \( k_i \), which are set to the value of \( \lambda_i \) and zero respectively in our implementation. An iterative calculation of \( k_i \) is performed by the routine from line 8 to 14. The state transition diagram of each processing element \( pe_i \) is constructed to build a set of linear equations.

```
1: Get_Mean_Waiting_Time(bus, pe_List, stat_params)
2: begin
3:   for each pe_i \in pe_List do
4:     Get_{\lambda_i} and \mu_i(stat_params)
5:     \theta_i = \lambda_i
6:     k_i = 0
7:   end for
8:   while (not all k_i are stable) do
9:     for each pe_i \in \{pe_1, \ldots, pe_{N-2}\} do
10:        Construct_State_Transition_Diagram(st_diag)
11:        Get_{k_i}(st_diag, \theta)
12:        Get_{\theta_i}(k_i)
13:     end for
14:   end while
15:   for each pe_i \in pe_List do
16:      w_i = k_i / \theta_i
17:   end for
18: end Get_Mean_Waiting_Time
```

Figure 3-4. The pseudo code describing the iterative procedure of the base queuing model.
Afterwards, the new value of $\theta_i$ and $k_i$ are obtained by Equation (3-1) in the current iteration (line 10 and 11). If differences between all $k_i$ at the current iteration and those of the previous iteration fall into a certain threshold, i.e. a predefined very small number close to zero, we quit the iteration and get $w_i$ using Equation (3-2). Otherwise, the iteration continues until all $k_i$ become stable. We used a linear program package [23] to solve obtained linear systems and found that the solutions are converged within less than 10 iterations in all cases, which took much less than the trace-driven simulation.

3.3.2 Extension to Processor Bus

The base queuing model of the previous section assumes a continuous system where the bus request can be served at any instance of time. In reality, however, bus arbitration is performed at discrete sampling points (i.e. clock edges) among all bus requests accumulated so far. If we assume that there occurs only one event, either request arrival or service completion, in each clock period, the base queuing model may be used as an approximate model. This assumption is suitable for the I/O bus case where bus requests are infrequent compared with the service time and the service time is relatively large. However, several events are very likely to occur during a single clock period in our case. Thus, we modify the base queuing model to allow simultaneous events. The state transition rate of a transition arc should be replaced by the state transition probability within a clock period.
As explained in the previous section, the state of the entire bus system is approximated by a set of state transition diagrams drawn for each processing element. To draw each state transition diagram, we divide the processing elements into three groups: the processing element of interest, ones with a higher priority and the others with a low priority. In this approximate model, an event is defined by a single increment or decrement for each coordinate of the state \((n_i, n_h, n_l, s)\). There are two kinds of events, which are the bus requests from three priority groups and the completion of current bus access. Table 3-1 and Table 3-2 show all possible state transitions in our compromised model. We classify the transitions into two types, *Type 1* and *Type 2*, depending on how many events occur during the next clock cycle. Note that we do not allow the state transition from \((n_i, n_h, n_l, s)\) to \((n_i, n_h+2, n_l, s)\) in the compromised model.

For the computation of the transition probabilities, we define two parameters \(d_h\) and \(d_l\) as the probabilities that at least one processing element issues a new request from \(\Phi_h\) and \(\Phi_l\) respectively:

\[
d_h = 1 - (1 - \gamma_h)^{N_h - n_h}, \quad d_l = 1 - (1 - \gamma_l)^{N_l - n_l}.
\]  

(3-9)

In the case that a processing element in \(\Phi_h\) is using a bus, the transition probability from \((n_i, n_h, n_l, s)\) to \((n_i, n_h+1, n_l+1, s)\) becomes \((1-\lambda_i) \cdot d_h \cdot d_f \cdot (1 - \gamma_h)\).
### Table 3-1. Type 1 transitions from the state \((n_i,n_h,n_s)\) to other states.

<table>
<thead>
<tr>
<th>The next state</th>
<th>The value of (s) in the current state</th>
<th>The transition rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>((n_i+1,n_h,n_s))</td>
<td>idle</td>
<td>(\lambda_i d_i (1-d_i)(1-d_i))</td>
</tr>
<tr>
<td></td>
<td>(h)</td>
<td>(\lambda_i d_i (1-d_i)(1-d_i))</td>
</tr>
<tr>
<td></td>
<td>(l)</td>
<td>(\lambda_i d_i (1-d_i)(1-d_i))</td>
</tr>
<tr>
<td>((n_i,n_h+1,n_s))</td>
<td>idle</td>
<td>((1-\lambda_i) d_i \mu_i)</td>
</tr>
<tr>
<td></td>
<td>(i)</td>
<td>(d_i \mu_i)</td>
</tr>
<tr>
<td></td>
<td>(h)</td>
<td>((1-\lambda_i) d_i \mu_i)</td>
</tr>
<tr>
<td></td>
<td>(l)</td>
<td>((1-\lambda_i) d_i \mu_i)</td>
</tr>
<tr>
<td>((n_i-1,n_h,n_s))</td>
<td>idle</td>
<td>((1-\lambda_i) d_i \mu_i)</td>
</tr>
<tr>
<td></td>
<td>(h)</td>
<td>((1-\lambda_i) d_i \mu_i)</td>
</tr>
<tr>
<td></td>
<td>(l)</td>
<td>((1-\lambda_i) d_i \mu_i)</td>
</tr>
</tbody>
</table>

### Table 3-2. Type 2 transitions from the state \((n_i,n_h,n_s)\) to other states.

<table>
<thead>
<tr>
<th>The next state</th>
<th>The value of (s) in the current state</th>
<th>The transition rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>((n_i+1,n_h+1,n_s))</td>
<td>idle</td>
<td>(\lambda_i d_i (1-d_i))</td>
</tr>
<tr>
<td></td>
<td>(h)</td>
<td>(\lambda_i d_i (1-d_i))</td>
</tr>
<tr>
<td></td>
<td>(l)</td>
<td>(\lambda_i d_i (1-d_i))</td>
</tr>
<tr>
<td>((n_i,n_h+1+1,n_s))</td>
<td>idle</td>
<td>((1-\lambda_i) d_i \mu_i)</td>
</tr>
<tr>
<td></td>
<td>(i)</td>
<td>(d_i \mu_i)</td>
</tr>
<tr>
<td></td>
<td>(h)</td>
<td>((1-\lambda_i) d_i \mu_i)</td>
</tr>
<tr>
<td></td>
<td>(l)</td>
<td>((1-\lambda_i) d_i \mu_i)</td>
</tr>
<tr>
<td>((n_i+1,n_h,n_s))</td>
<td>idle</td>
<td>((1-\lambda_i) d_i \mu_i)</td>
</tr>
<tr>
<td></td>
<td>(h)</td>
<td>((1-\lambda_i) d_i \mu_i)</td>
</tr>
<tr>
<td></td>
<td>(l)</td>
<td>((1-\lambda_i) d_i \mu_i)</td>
</tr>
<tr>
<td>((n_i,n_h+1,1,s))</td>
<td>idle</td>
<td>((1-\lambda_i) d_i \mu_i)</td>
</tr>
<tr>
<td></td>
<td>(i)</td>
<td>(d_i \mu_i)</td>
</tr>
<tr>
<td></td>
<td>(h)</td>
<td>((1-\lambda_i) d_i \mu_i)</td>
</tr>
<tr>
<td></td>
<td>(l)</td>
<td>((1-\lambda_i) d_i \mu_i)</td>
</tr>
<tr>
<td>((n_i+1,n_h,n_s))</td>
<td>idle</td>
<td>((1-\lambda_i) d_i \mu_i)</td>
</tr>
<tr>
<td></td>
<td>(h)</td>
<td>((1-\lambda_i) d_i \mu_i)</td>
</tr>
<tr>
<td></td>
<td>(l)</td>
<td>((1-\lambda_i) d_i \mu_i)</td>
</tr>
</tbody>
</table>
In order to validate the accuracy of our compromised model, we compare it with a more general but complicated model to deal with $K$ simultaneous events. In this model we also allow a certain priority group to request more than one simultaneous event. Then the transition from the state $(n_i, n_h, n_l, s)$ to the state $(n_i^*, n_h^*, n_l^*, s^*)$ is valid as long as the following constraints are satisfied:

$$0 \leq n_i^* \leq 1, \quad n_h - 1 \leq n_h^* \leq N_h, \quad n_l - 1 \leq n_l^* \leq N_l,$$

$$\left| n_i - n_i^* \right| + \left| n_h - n_h^* \right| + \left| n_l - n_l^* \right| \leq K.$$  \hspace{1cm} (3-10)

The rate of the transition from $(n_i, n_h, n_l, s)$ to $(n_i^*, n_h^*, n_l^*, s^*)$, $TR$, can be expressed as the product of three terms, i.e., $TR = tr_i \cdot tr_h \cdot tr_l$, where $tr_i$, $tr_h$, and $tr_l$ are the contributions by the events from the three priority groups respectively. The equations for the terms are given in Equation (3-11), Equation (3-12), and Equation (3-13). $tr_i$ indicates the transition rate from $n_i$ to $n_i^*$ ignoring other processing elements. The equations for $tr_h$ and $tr_l$ are complicated depending on which case the next state might belong to:

- The case where the number of processing elements reaches $N_h$ or $N_l$;
- The case where the priority group is not granted a bus and new requests occur;
- The case where the priority group is granted a bus and new requests occur;
- The case where the service for the request is completed.
Chapter 3  Queuing Model of On-Chip Bus Architectures

\[
\begin{align*}
\mu_i & \quad \text{if } n_i = 1 \text{ and } n^*_i = 0, \\
1 - \lambda_i & \quad \text{if } n_i = 0 \text{ and } n^*_i = 0, \\
1 - \mu_i & \quad \text{if } n_i = 1 \text{ and } n^*_i = 1 \text{ and } s = 'l', \\
\mu_i & \quad \text{if } n_i = 0 \text{ and } n^*_i = 1, \\
1 & \quad \text{Otherwise.}
\end{align*}
\]  
(3-11)

\[
\begin{align*}
(1 - \beta_h) \left( \frac{N_h - n_h}{n^*_h - n_h} \right) & \cdot \gamma_h^{n^*_h - n_h} \cdot (1 - \gamma_h)^{N_h - n^*_h} \quad \text{if } n^*_h = N_h \text{ and } s = 'h', \\
\left( \frac{N_h - n_h}{n^*_h - n_h} \right) & \cdot \gamma_h^{n^*_h - n_h} \cdot (1 - \gamma_h)^{N_h - n^*_h} \quad \text{if } n_h \leq n^*_h \leq N_h \text{ and } s \neq 'h', \\
(1 - \beta_h) \left( \frac{N_h - n_h}{n^*_h - n_h + 1} \right) & \cdot \gamma_h^{n^*_h - n_h + 1} \cdot (1 - \gamma_h)^{N_h - n^*_h + 1} + \beta_h \left( \frac{N_h - n_h}{n^*_h - n_h + 1} \right) \cdot \gamma_h^{n^*_h - n_h + 1} \cdot (1 - \gamma_h)^{N_h - n^*_h + 1} \quad \text{if } n_h \leq n^*_h < N_h \text{ and } s = 'h', \\
\beta_h \cdot \gamma_h^{N_h - n_h} & \quad \text{if } n^*_h = n_h - 1, \\
1 & \quad \text{Otherwise.}
\end{align*}
\]  
(3-12)

\[
\begin{align*}
(1 - \beta_l) \left( \frac{N_l - n_l}{n^*_l - n_l} \right) & \cdot \gamma_l^{n^*_l - n_l} \cdot (1 - \gamma_l)^{N_l - n^*_l} \quad \text{if } n^*_l = N_l \text{ and } s = 'l', \\
\left( \frac{N_l - n_l}{n^*_l - n_l} \right) & \cdot \gamma_l^{n^*_l - n_l} \cdot (1 - \gamma_l)^{N_l - n^*_l} \quad \text{if } n_l \leq n^*_l \leq N_l \text{ and } s \neq 'l', \\
\beta_l \left( \frac{N_l - n_l}{n^*_l - n_l + 1} \right) & \cdot \gamma_l^{n^*_l - n_l + 1} \cdot (1 - \gamma_l)^{N_l - n^*_l + 1} \quad \text{if } n_l \leq n^*_l < N_l \text{ and } s \neq 'l' \text{ and } s^* = 'l', \\
\beta_l \cdot \gamma_l^{N_l - n_l} & \quad \text{if } n^*_l = n_l - 1, \\
1 & \quad \text{Otherwise.}
\end{align*}
\]  
(3-13)
Figure 3-5. The average estimation error of the compromised queuing model compared to the general queuing models with the various numbers of simultaneous events.

Figure 3-6. The variations of estimation error of the compromised queuing model compared to the general queuing models with the various numbers of simultaneous events.
Chapter 3  Queuing Model of On-Chip Bus Architectures

Figure 3-7. The execution time to solve the compromised queuing model and the general queuing models with the various numbers of simultaneous events.

Comparisons between the base queuing model, our compromised model, and the general model are shown in Figure 3-5, Figure 3-6, and Figure 3-7. We use single bus systems and randomly generated memory traces in the comparisons. The general model is configured with the various numbers of maximum simultaneous events, 2 to 4. The estimation errors are computed against the trace-driven simulation results. More detailed explanation on the experimental environment is refereed to Section 4.4.1.

Figure 3-5 shows that both our compromised model and the general model achieve the significant accuracy improvement compared with the base model. However, even though the general model tends to be more accurate with more simultaneous events, the accuracy improvement is not remarkable. Furthermore, as shown in Figure 3-6, it is observed that no clear distinction exists for the variations
of the estimation errors between the compromised and the general models. Figure 3-7 represents the execution time of linear program solver. Execution times for solving the models are about the same because the problem size by the queuing model is not dependent on the number of transitions but dependent on the number of states. But the modeling complexity grows exponentially as the number of simultaneous events grows. Considering those observations, we confirm that our compromised model can be used effectively.

3.4 Model of Round-Robin based Bus

The queuing model of a fixed priority bus serves as the base model for other arbitration schemes. In this section, we present the queuing model for a bus with a round-robin arbitration scheme. The round-robin arbitration is commonly used in on/off-chip bus standards as not only a hybrid arbitration scheme, incorporating with fixed priority or TDMA arbitration [21][22], but also an independent arbitration policy by itself [55][56].

A round-robin bus guarantees avoidance of starvation among processing elements and allows any unused time slot to be allocated to a master whose round-robin turn is later but who is ready now [55]. Contrary to the fixed priority scheme, a round-robin scheme varies dynamically the priority of each processing element between the highest and the lowest depending on the position of the processing element and the distance from the most recent processing element accessing a bus.
in the circular bus grant order. Processing elements are granted a bus in the order of their indices wrapped around.

Two implementations of round-robin arbitration are common as illustrated in Figure 3-8. A difference between them can be found in the way of changing the processing element with the highest priority. In the case of Implementation 1 in Figure 3-8(a), if arbitration is to be made every instant of time $T_n$ ($n=0,1,\ldots$) and no bus request had arrived at $T_i$, the priorities made at $T_0$ should be kept. There is no
change in priorities as long as no bus request arrives. Hereafter, it can be changed due to actual bus arbitration as it happens at $T_2$. On the other hand, in the case of Implementation 2 in Figure 3-8(b), priorities can be varied even if there is no bus request accumulated until $T_1$. The next processing element gets the highest priority such that the priorities change regularly every instant of time.

To construct a state transition diagram for the round-robin system, we first consider Implementation 1 and decompose a round-robin system into multiple fixed priority systems as shown in Figure 3-9. We define $BS_{FP}(N,i)$ as the fixed priority based bus system with $N$ processing elements $(PE_0, ..., PE_{N-1})$ and the priorities given in the sequence of $(PE_i, ..., PE_{N-1}, PE_0, ..., PE_{i-1})$. $PE_i$ has the highest priority. Next, we define $BS_{RR}(N)$ as the round-robin based bus system with $N$ processing elements $(PE_0, ..., PE_{N-1})$. Therefore, we have

$$BS_{RR}(N) = \{bs_{RR}(i) \mid bs_{RR}(i) = BS_{FP}(N, i) \} \forall i \in \{0, ..., N-1\}.$$
A hierarchical state $bs_{RR}(j)$ is entered when $PE_{j-1}$ completes a bus access. Therefore, the rate of a state transition from $bs_{RR}(i)$ to $bs_{RR}(j)$ is the product of the service rate of and the actual bus access rate of $PE_{j-1}$ in the fixed priority system inside $bs_{RR}(i)$. If we denote $\theta_{j-1}$ as the actual bus access rate of $PE_{j-1}$ inside $bs_{RR}(i)$, it can be computed by solving the fixed priority based system using the queuing model explained in the previous section. In fact, $\theta_{j-1}$ is equal to $\theta_{j}$ of the previous section if the priority list is identically set. Since the service rate of $PE_{j-1}$ is consistent throughout the execution, the state transition rate $\delta_{j}$ from $bs_{RR}(i)$ to $bs_{RR}(j)$ becomes

$$\delta_{i,j} = \theta_{i,j-1} \cdot \mu_{j-1}.$$  (3-14)

In the case of Implementation 2, the state transition rate in Equation (3-14) is needed to be slightly modified to consider the regular change of priorities without regard to presence of bus requests. If $j$ is next to $i$ in a given round-robin sequence, a state transition from $bs_{RR}(i)$ to $bs_{RR}(j)$ can take place even if no bus request is present. Therefore, we get a state transition rate of Implementation 2, $\delta_{i,j}^*$, as follows:

$$\delta_{i,j}^* = \theta_{i,j-1} \cdot \mu_{j-1} + \prod_{k=0}^{N-1} (1 - \theta_{i,k}).$$  (3-15)

Once all transition rates are determined, we obtain the steady state probability of each hierarchical state using an additional requirement of $\sum_{k=0}^{N-1} P(bs_{FP}(k)) = 1$.
as similar to Section 3.3.1. Now we are ready to compute the expected wait time $w_{RR,i}$ for a bus access of $PE_i$ on the round-robin based system. It is the weighted sum of the expected wait time $w_{k,i}$ of $PE_i$ in each hierarchical state $bs_{RR}(k)$. The weighted sum should be able to explain that the average wait time of a processing element is proportional to: (a) a probability that the processing element is in a state of interest and (b) an average bus request rate while the processing element stays in the state. Finally, we arrive at the following equation:

$$w_{RR,i} = \frac{\sum_{k=0}^{N-1} \left( P(bs_{FP}(k)) \cdot \theta_{k,i} \cdot w_{k,i} \right)}{\sum_{k=0}^{N-1} \left( P(bs_{FP}(k)) \cdot \theta_{k,i} \right)}.$$ (3-16)

### 3.5 Model of Two-Level TDMA based Bus

We also use a hierarchical state diagram to model more complicated arbitration policies: modeling of a two-level TDMA based bus is explained in this section. It consists of two arbitration schemes, primary TDMA and secondary round-robin. The first level of arbitration uses a timing wheel where each slot is statically reserved for a unique master. A processing element with heavy communication requirements may reserve more than one slot. Although the TDMA based architecture guarantees a fixed bus bandwidth for each processing element, no bus request from the processing element associated with the current slot means the waste of bus bandwidth. In order to prevent the waste of this unused slot, another
processing element may be granted the bus by round-robin arbitration during the slot.

A two-level TDMA system can also be treated as the composition of multiple fixed priority systems similarly to the round-robin based system discussed in the previous section. Consider the example system that has four processing elements \((PE_0, \ldots, PE_3)\). When the total bandwidth of a bus is normalized to 1, \(PE_i\) is assigned a bandwidth \(bw(i)\) less than 1 and the sum of the bandwidths of all processing elements becomes 1. Suppose that \(PE_0\) is assigned the current TDMA slot and \(PE_i\) has the highest priority in the round-robin order. The bus grant order for round-robin arbitration is again assumed to be the order of processing element indices. Therefore, the priority at the current time slot will be \((0,1,2,3)\), i.e. \(PE_0\) has the highest priority and \(PE_1\) becomes the second, and so on.

More generally, we define an assignment sequence of time slots as follows:

\[
TS(N) = \{ts(i) \mid \sum_{i=0}^{N-1} bw(i) = 1 \} \quad \forall i \in \{0, \ldots, N-1\},
\]

where \(N\) processing elements are assigned and \(ts(i)\) is a set of time slots that \(PE_i\) is assigned. We also define the two-level TDMA based bus system as:

\[
BS_{TT}(N) = \{bs_{TT}(i, j) \mid bs_{TT}(i, j) = ts(i) \cup BS_{FP}(N-1, j) \} \quad \forall i, j \in \{0, \ldots, N-1\},
\]

where \(bs_{TT}(i, j)\) is the hierarchical state in which \(PE_i\) is assigned the current time slot and \(PE_i\) has the highest priority in the priority based system with \(N-1\).
processing elements except for $PE_i$. Therefore, the overall priority order in $bs_{TT}(i, j)$ becomes $(PE_i, PE_j, \ldots, PE_{N-1}, PE_{0}, \ldots, PE_{i-1})$.

**Figure 3-10.** (a) An example of TDMA slot assignment and (b) the state transitions from $bs_{TT}(0, 1)$ according to the slot assignment in (a).

Figure 3-10(b) shows the example state transition from $bs_{TT}(0, 1)$ for a given fixed 16-slot assignment of the TDMA protocol as shown in Figure 3-10(a). If $PE_0$ is granted a bus in the state $bs_{TT}(0, 1)$, the next state is one of $bs_{TT}(0, 1)$, $bs_{TT}(1, 2)$, $bs_{TT}(2, 1)$, and $bs_{TT}(3, 1)$ depending upon which processing element gets the next TDMA slot. Overall 12 transitions can be drawn from $bs_{TT}(0, 1)$ as shown in Figure 3-10(b). But some destination states are duplicated in the figure. For instance, the transition from $bs_{TT}(0, 1)$ to $bs_{TT}(1, 2)$ can occur by granting the bus to either $PE_0$ or $PE_1$. On the other hand, some state transitions are not allowed in the given slot assignment. For example, when $PE_0$ is assigned the current slot in Figure 3-10(b), the next slot can be allocated only to $PE_1$ and $PE_2$ so that no transitions to $bs_{TT}(0, x)$
Chapter 3  Queuing Model of On-Chip Bus Architectures

and to $bs_{TT}(3, x)$ can be made. Those infeasible transitions are represented as the dashed lines in Figure 3-10(b). In general, for given $N$ processing elements, there are state transitions less than $N\cdot(N-1)$ from a certain state.

We define $b_{i,m}$ as the probability that $PE_m$ can be assigned the next TDMA slot when $PE_i$ is assigned the current slot. In the case of Figure 3-10(a), $b_{0,1}$ and $b_{0,2}$ become 0.67 and 0.33 respectively. Both $b_{0,0}$ and $b_{0,3}$ are zero. Suppose that the destination state is $bs_{TT}(m,n)$ when $PE_k$ is granted a bus in $bs_{TT}(i,j)$. Then the transition rate from $bs_{TT}(i,j)$ to $bs_{TT}(m,n)$, $\delta_{(i,j)\rightarrow(m,n)}$, becomes the product of the actual bus request rate of $PE_k$ in $bs_{TT}(i,j)$, the service rate of $PE_k$, and $b_{i,m}$ so that we get the following equation,

$$\delta_{(i,j)\rightarrow(m,n)} = b_{i,m} \cdot \theta_{k,(i,j)} \cdot \mu_k,$$  \hspace{1cm} (3-17)

where $\theta_{k,(i,j)}$ is the actual bus request rate of $PE_k$ in $bs_{TT}(i,j)$ and can be obtained by solving the fixed priority system where $PE_i$ and $PE_j$ have the highest and the second priorities respectively as explained in Section 3.3. With these state transitions rates and an additional requirement of $\sum P(bs_{TT}(i,j)) = 1$, we obtain the steady state probability of each $bs_{TT}(i,j)$. To compute the expected wait time for each processing element, a similar equation as Equation (3-16) can be used.
3.6 Chapter Summary

In this chapter, we presented queuing models of bus architectures considering three commonly used arbitration schemes: fixed priority, round-robin, and two-level TDMA. The queuing model of a fixed priority bus is based on a simple I/O bus. Since a processor-memory system bus is different from an I/O bus in terms of frequency of bus access requests, the base model of I/O bus was extended to consider simultaneous events including access request to bus and completion of memory access. Then, other arbitration schemes, round-robin and two-level TDMA, were modeled as a hierarchical system that is represented as multiple fixed priority systems and stays in certain one at any time instance. In the next chapter, we will show that the proposed queuing models deliver significant improvement in accuracy of performance estimation as well as advantage in time efficiency comparable to that of simulation approach under various architecture configurations.
Chapter 4

Performance Estimation using Static Schedule

4.1 Estimation of Single Bus

This section explains how the task schedule information is used in our estimation method. A simple statistical modeling of the bus requests from a processing element assumes that the bus requests are distributed evenly throughout the whole execution duration of an application. This assumption is one of the main sources of the inaccuracy of simple static statistical modeling. Since we assume that the schedule of function blocks is pre-determined, the pattern of bus requests is
determined statically as shown in Figure 4-1. Note that the initial schedule is made without considering the wait time for bus access. We divide the schedule into several time slots in such a way that all processing elements maintain their bus request patterns during each time slot. Then, we apply the proposed queuing analysis in each time slot starting from the beginning of the schedule. The shaded regions in the figure indicate the remaining sections for the static estimation.

Figure 4-1. (a) An example schedule of \(PE_0\), \(PE_1\), and \(PE_2\) and its corresponding queuing model, (b) new queuing model after function block \(C\) is finished at \(T_1\), and (c) another queuing model after function block \(B\) is finished at \(T_2\).

During the first time slot, three function blocks \(A\), \(B\), and \(C\) are executed concurrently on \(PE_0\), \(PE_1\), and \(PE_2\) respectively. To evaluate the expected wait delay for bus access from each processing element, a queuing system is constructed.
Chapter 4  Performance Estimation using Static Schedule

as shown in Figure 4-1(a). From the proposed queuing analysis, we compute how much the initial schedule length of function blocks is extended due to bus contention. Suppose that the function block $C$ on $PE_2$ is finished first at $T_i$. Then we consider the next time slot two function blocks $A$ and $B$ are accessing a bus as shown in Figure 4-1(b) and construct another queuing model with $PE_0$ and $PE_1$. This time slot is also lengthened until the function $B$ is completed at $T_2$. After $T_2$, another queuing model with $PE_0$ and $PE_2$ is made for the evaluation of the remaining schedule. This evaluation process is repeated until the queuing model examines all of the scheduled function blocks.

The overall algorithm of the estimation technique for a single bus is described in Figure 4-2. The procedure **Estimate_Single_Bus** has five inputs. $BUS$ represents the data structure of a single bus. $MT$ is the memory traces of all processing elements and $ORI_SCHED$ is the initial schedule. $PL$ is a set of processing elements and $FL$ is a set of function blocks to be used for static estimation. The output, $EVAL_SCHED$, of this procedure is the updated schedule after considering all bus conflicts and other overheads if any.
Figure 4-2. The schedule-aware performance estimation algorithm for a single bus.

The main procedure of the queuing analysis is **Estimate_End_Time**. Before the queuing analysis, we first determine which processing elements request a bus by calling the procedure **Get_Current_Fb** and compute the statistical parameters by calling the procedure **Get_Stat_Params**. We compute two statistical parameters of each function block from the memory traces $MT$: they are the memory access rate $\lambda$ with no bus conflicts and the mean service time $S$. The memory access rate during the execution of the function block $FB$ on the processing element $PE$ is formulated as follows:
where $M_{FB,PE}$ is the total memory access counts and $EXE_{FB,PE}$ is the execution time of the function block $FB$ on the processing element $PE$. When computing the mean service time, we consider the different burst transfer size according to the memory access type. For example, code memory access is usually the burst access of which the size equals to the cache line size, whereas data memory access may have various burst lengths. Therefore, we define a set of memory access types $AT$ with three types: code memory access $AccCode$, data memory access $AccData$, and shared memory access $AccShared$,

$$AT = \{AccCode, AccData, AccShared\}.$$  

For each set of access types, we further define a set of burst transfer types $BT$ according to the burst transfer length:

$$BT = \{single, four-beat, eight-beat, random\}.$$  

There may be more burst transfer types in general. However, for simplicity, we assume only 4 types of burst transfers, which are used in the ARM720T processor [57]. Then, the mean service time for the function block $FB$ on the processing element $PE$ is computed as

$$S_{FB,PE} = \sum_{at \in AT} \left( \sum_{bt \in BT} S_{at,bt,PE} \cdot \frac{M_{at,bt,FB,PE}}{M_{FB,PE}} \right),$$  

(4-2)
where $S_{at, bt, PE}$ and $M_{at, bt, FB, PE}$ are the service time including the bus overhead as well as the memory access time and the burst transfer counts of the type $bt$ in the memory access type $at$ for the function block $FB$ allocated on the processing element $PE$ respectively.

The final procedure **Update Schedule** modifies the initial schedule to obtain the updated schedule, $EVAL\_SCHED$, after the current time slot. From the updated schedule, we define the next time slot and go back to the main iteration body until all function blocks are considered. In the case that the number of function blocks is huge or the schedule length is very long, the time complexity of our proposed technique approaches to that of the trace-driven simulation due to too frequent invocation of LP solver. To reduce this problem, neighboring function blocks can be clustered into a group, which is discussed in the next section.

### 4.2 Extension to Multiple Bus Systems

Communications across buses are achieved via a bus bridge in multiple bus systems. A bus bridge plays both roles of a processing element and a memory as displayed in Figure 4-3. We assume for simple analysis that no communication passes through more than 3 buses.

Figure 4-3 shows how the bridge is modeled when a processing element on the bus $src$ accesses the memory $Mem$ on the bus $dest$. The request rate $\lambda_{src}$ of the
processing element $PE$ is reflected to the bus $dest$ by the request rate $\lambda_{dest}$ of the bridge. To compute $\lambda_{dest}$, we have to know the expected waiting time $w_{src}$ of the processing element $PE$ on the bus $src$. At the same time, the bridge looks like a memory from the bus $src$ point of view. Therefore, the service rate $\mu_{src}$ of the bridge should be computed. Let $w_{dest}$ be the expected waiting time of the bridge on the bus $dest$. Then, $\lambda_{dest}$ and $\mu_{src}$ are computed as follows:

$$
\frac{1}{\lambda_{dest}} = \frac{1}{\lambda_{src}} + w_{src} + O_{bridge}, \quad \frac{1}{\mu_{src}} = w_{dest} + O_{bridge} + \frac{1}{\mu_{dest}} \quad (4-3)
$$

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

\[\begin{array}{c}
\lambda_{src} \\
\mu_{src} \\
\lambda_{dest} \\
\mu_{dest}
\end{array}\]

**Figure 4-3. The modeling of communication via a bus bridge.**

Therefore, the overall estimation algorithm in Figure 4-2 can be extended as described in Figure 4-4. Contrary to the single bus estimation of Figure 4-2, we need an additional data structure to represent a network of multiple buses, which
includes lists of multiple buses and bridges as BUS and BRIDGE respectively. An
loop from line 19 to 42 is newly added to the most outer loop starting at line 9,
which implements the estimation of multiple bus system.

Referring to Figure 4-3, in order to get the new values of parameters involved in
bus src such as \( w_{src} \), the parameters from other buses such as \( \lambda_{dest} \), should be kept,
and vice versa. In practice, many buses might be involved as destinations like bus
dest in Figure 4-3 to get parameters of a certain bus that is a source. It means that
communication traffics of a certain bus can be contributed by bus accesses initiated
by other buses. To deal with those more generally, we iteratively execute the loop
until bridge-related parameters explained above are converged. The estimation
starting from line 19 is performed through two steps, which are from line 21 to line
26 and from line 27 to line 36 respectively.

The procedure **construct_estimation_subset** constructs a set of processing
elements accessing bus at line 21 and line 27 to execute their associated function
blocks according to the given schedule \( ORI\_SCHED \). In the procedure
**initialize_bridge_as_pe**, communication traffics spanning bridges are collected
and, then, the statistical variables of bridges are extracted to use them in the
queuing analysis, where a bridge plays a role of processing element. Therefore,
bridges as processing elements can be transparently delivered to the procedure
**Estimate_End_Time** by the argument \( PL \). The second for loop from line 27 to
line 37 is identical to the first one except for last six lines to judge whether a
convergence is made or not. In the routine, the average wait time of a bridge as a
processing element is compared with that of previous iteration of loop of line 19. If
they are different from each other within certain threshold value, a convergence
was made and no more iteration is required. If current iteration failed to make a
convergence, the next iteration is invoked to run with the average wait time of each
bridge that has been replaced with the newer one. In our implementation, the
threshold `convergence_threshold` is set to 0.001, *i.e.* the difference of average
wait time between consecutive iterations should less than 0.1%.
Chapter 4  Performance Estimation using Static Schedule

Figure 4-4. The schedule-aware performance estimation algorithm for multiple-bus architecture.
4.3 Time Complexity

As explained in the previous section, our proposed technique progresses dividing the entire schedule into several time slots in which separate queuing models are constructed and the proposed queuing analysis is applied to. Therefore, the time complexity of the proposed technique depends on the product of the time complexity of the queuing analysis and the number of time slots. First, we consider the time complexity of the queuing analysis.

<table>
<thead>
<tr>
<th>Fixed priority</th>
<th>Number of processing elements</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum number of states</td>
<td>5</td>
<td>21</td>
<td>37</td>
<td>53</td>
<td>69</td>
<td></td>
</tr>
<tr>
<td>Maximum number of states</td>
<td>5</td>
<td>21</td>
<td>47</td>
<td>83</td>
<td>129</td>
<td></td>
</tr>
<tr>
<td>CPU time (seconds)</td>
<td>0.04</td>
<td>0.11</td>
<td>0.34</td>
<td>0.92</td>
<td>2.68</td>
<td></td>
</tr>
</tbody>
</table>

| Round-robin | CPU time (seconds) | 0.04 | 0.21 | 1.22 | 5.62 | 21.05 |
| Two-level TDMA | CPU time (seconds) | 0.04 | 0.44 | 5.16 | 32.16 | 167.59 |

Table 4-1 shows the run time of the proposed estimation technique and the maximum and the minimum number of states varying the number of processing elements on Xeon 2.8GHz workstation running Linux. We used a GNU Scientific Library (GSL) to solve the linear equations of the proposed method [23]. In the case of the fixed priority base system, for each processing element $PE_i$, the number
of states \(\{(n_i, n_h, n_l, s)\}\) is \(O(N^2)\) where \(N\) is the total number of processing elements. The number of states depends on the priority of the processing element of interest. Since the time complexity of the LP solver is pseudo-polynomial, the overall time complexity is also pseudo-polynomial.

As for the round-robin based system, since there exist \(N\) priority lists for \(N\) processing elements, the complexity becomes roughly \(N\) times larger than that of the fixed priority based system. On the other hand, since the two-level TDMA based system with \(N\) processing elements may have up to \(N(N-1)\) priority lists, its complexity is roughly \(N(N-1)\) times larger than the fixed priority case. Note that all three cases still have the pseudo-polynomial time complexity although the overall complexity increase as the arbitration scheme becomes complicated. Table 4-1 confirms that the proposed technique has the acceptable complexity for fast design space exploration.

### 4.3.1 Comparison with Trace-Driven Simulation

While the time complexity of the proposed technique depends on the number of processing elements as discussed above, that of the trace-driven simulation depends on the trace size as well as the number of processing elements. Considering those parameters, comparisons are made quantitatively by measuring the execution time as shown in Figure 4-5 and Figure 4-6. The number of processing elements is varied from 2 to 16. The average schedule length of each
processing element is also varied from $10^5$ cycles to $10^7$ cycles. In Figure 4-5, the average bus request rate for each processing element is set to about 0.12, which is a typical value for multimedia applications. On the other hand, highly intensive memory access rate, 0.17, is applied to the result in Figure 4-6.

The time complexity of the proposed technique increases faster than the trace-driven simulation as the number of processing elements increases. It explains why the time complexity of the proposed technique is larger when the schedule length is short and the number of processing elements is 16 in Figure 4-5. However the execution time of the trace-driven simulation grows proportionally to the trace size while that of the proposed approach remains the same. Therefore the benefit of using the proposed technique grows as the length of the time slot increases. Such tendency appears significantly in the case of higher memory access rate as depicted in Figure 4-6. The observation above implies that the proposed estimation technique can be more effective even in future SoC designs since much more communication traffics would incur.
Chapter 4  Performance Estimation using Static Schedule

Figure 4-5. The comparison of time complexity between the proposed technique and trace-driven simulation using typical memory access rate: 0.12.

Figure 4-6. The comparison of time complexity between the proposed technique and trace-driven simulation using highly intensive memory access rate: 0.17.
4.3.2 Effects of Schedule Complexity

The time complexity of the proposed technique is also proportional to the number of time slots. Since each time slot is defined when a function block finishes its execution, the number of time slots is about the same as the number of scheduled function block invocations. In many video applications, the number of function block invocations can be very large due to the different execution rates of function blocks. Figure 4-7 shows an H.263 decoder specification example that is mapped to two processing elements: an ARM9 processor and a dedicated hardware for IDCT as specified in Figure 4-7(a) and Figure 4-7(b) respectively. To decode one macro block, one invocation of gray function blocks and four invocations of black function blocks are required. To decode one QCIF-formatted frame that consists of 99 macro blocks, the schedule contains the total 6968 invocations of the function blocks.

Figure 4-8 shows the naïve application of the proposed technique is worse than the trace-driven simulation since the average granularity of the function blocks is too small. Therefore we group the function blocks that are repeatedly executed in the same sequence. Such grouping also reduces the effective number of time slots. In this example, we group the function blocks that decode one macro block and apply the proposed estimation technique to the first invocation of the group. A significant efficiency gain, about 47 times, is obtained by grouping the function blocks as shown in the figure.
Chapter 4  Performance Estimation using Static Schedule

Figure 4-7. The specification of H.263 Decoder and the mapping of function blocks to (a) an ARM9 processor and (b) a dedicated IDCT hardware. The groups in (a) and (b) are executed 99 times during decoding one frame.

Figure 4-8. Grouping the function blocks in the schedule is required to enhance the efficiency of the proposed technique.
4.4 Experiments

4.4.1 Estimation Accuracy for Single Bus Systems

To investigate the accuracy of the proposed static estimation method over a wide variety of working conditions, the first set of experiments considers a single bus architecture varying number of processing elements, bus request rates, bus service rates (or memory access times), and burst lengths. Then we compared the estimation results with those obtained from the trace-driven simulation. Our trace-driven simulator adjusts the time stamps of trace data by accurately modeling the communication architecture that includes buses and memories. More precisely, in these experiments, the bus model used in our trace-driven simulator consists of 4 phases: bus-arbitration, start-address-drive, sequential-burst-transfer, and last-data-drive.

Figure 4-9 shows the burst transfer of four words complying with this bus model. The part tagged with ‘A’ means the waiting time of a processing element due to bus conflicts. The parts B, C, and E are the bus protocol specific overheads and are all set to 1 cycle in this experiment. The part D consists of more than one word accesses to the memory. The memory is configured with two parameters, which are
the initialization cycle for burst transfer and the access cycle taken for a single word during the burst transfer. For example, they are set to 1 and 2 clock cycles respectively in Figure 4-9. It is assumed that other control signals are synchronized with the address bus and are not shown here for simplicity.

A template example system for our first set of experiments is described in Figure 4-10. \( N \) processing elements are selected and all processing elements are busy during the same schedule length, 10,000 cycles. In order to maximize bus conflicts, all processing elements are connected to a single bus as shown in Figure 4-10(b). The number of processing elements and the memory subsystem access time define the configuration points in this architecture template. We vary the number of

---

**Figure 4-9. The burst transfer of four words on our bus model.**

A: Waiting time for the use of a bus (dependent of bus conflicts)  
B: Bus arbitration  
C: Start address drive  
D: Sequential burst transfer (consists of burst initialisation and sequential accesses)  
E: Last data drive
processing elements from 2 to 10. The bus request rate $\lambda$ of each processing element is chosen randomly within the range from 0.05 to 0.2. For a selected bus request rate, we generated the memory traces from each processing element following the Poisson distribution (i.e. the exponentially distributed inter-arrival times between bus requests).

![Figure 4-10](image)

Figure 4-10. (a) The schedule of function blocks on $N$ processing elements ($PE_0$, $PE_1$, $PE_2$, $\ldots$, $PE_{N-1}$) and (b) a single bus architecture that consists of $N$ processing elements and a memory subsystem.

While generating the memory traces, we also randomized the burst lengths. As for the memory subsystem, the burst initialization overhead is always fixed to 1 cycle. The single word access time of each processing element is assigned in four different ways: the identical assignments of 1, 3, and 5 cycles to all processing elements and the random assignments, between 1, 3, and 5 cycles, to each processing element. In the case that $\lambda$ is 0.1 and memory access time is 1 cycle, the portion of the total memory access time over the entire execution time is about 0.36.
This portion is almost same as the case of an H.263 encoding algorithm, which will be discussed later.

Figure 4-11. The estimation accuracy according to the number of processing elements and the memory access time for each arbitration scheme.
Using these configuration points, 20 example systems were constructed for each of three arbitration schemes: fixed priority, round-robin, and two-level TDMA. We performed the proposed static analysis and the trace-driven simulation to compare the total execution time of the example systems. For each set of experiments, we generated the memory traces 10 times. Figure 4-11 shows the experimental results, which indicate the range of estimation errors of the completion times. It is observed that the estimation error of the proposed method does not exceed 4% in all cases. These experiments show the robustness of the proposed technique on various architecture configurations and arbitration schemes.

4.4.2 Comparison with Simpler Models

In this section, we show the improvement of our proposed model over two simpler estimation models: the base queuing model of an I/O bus reviewed in Section 4.1 and the intuitive analytical model assuming a fixed priority bus system. It is questionable whether there is a simpler static estimation method that is reasonably accurate. So, for a single bus that has \( N \) processing elements, we devised an intuitive equation on the expected waiting time of \( PE_i \), \( w_i \), by bus contention as shown in Equation (4-4).

\[
w_i = \frac{\sum_{j=0}^{i-1} m_j}{\text{exe}_i} \cdot \frac{S_i}{2} \cdot (i-1) + \frac{\sum_{j=i+1}^{N-1} m_j}{\text{exe}_i} \cdot \frac{S_i}{2},
\]

(4-4)
where $m_j$, $exe_j$, and $S_j$ are the total memory access time, the total execution time, and the mean service time of $PE_j$ respectively. At the right side of Equation (4-4), the first term is the expected waiting time due to the bus requests from the priority group $\Phi_h$ and the second term is the waiting time due to the current outstanding request of the priority group $\Phi_l$. The same experimental environment in Figure 4-10 is used again. We estimated the performance of 20 communication architecture configurations using the base model of an I/O bus and Equation (4-4). Each experiment was repeated 10 times with randomly generated memory traces. Figure 4-12(a), (b), and (c) summarize the estimation results by three estimation techniques in terms of the absolute error range compared with the trace-driven simulation. One bar graph for a given number of processing elements covers all kind of the memory access times mentioned in the previous section.

Although the average estimation error of the base queuing model does not exceed 6% as shown in Table 4-2, the variation of errors becomes larger rapidly as the number of processing elements increases. The intuitive equation does not produce the acceptable estimation accuracy on the variation of errors as well as the average error rate. On the other hand, the processor bus model shows the consistent estimation errors within the range of about 1.3% on average.
Figure 4-12. The comparison of estimation errors for the example systems of Figure 4-10 between (a) the base queuing model for an I/O bus, (b) the intuitive equation, and (c) the proposed model for the processor bus.
Table 4-2. Average estimation errors according to the estimation techniques.

<table>
<thead>
<tr>
<th>Number of processing elements</th>
<th>I/O bus model</th>
<th>Intuitive equation</th>
<th>Processor bus model</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1.4 %</td>
<td>16.6 %</td>
<td>1.3 %</td>
</tr>
<tr>
<td>4</td>
<td>2.4 %</td>
<td>26.3 %</td>
<td>1.1 %</td>
</tr>
<tr>
<td>6</td>
<td>4.4 %</td>
<td>22.6 %</td>
<td>0.5 %</td>
</tr>
<tr>
<td>8</td>
<td>5.8 %</td>
<td>18.5 %</td>
<td>0.6 %</td>
</tr>
<tr>
<td>10</td>
<td>5.9 %</td>
<td>14.0 %</td>
<td>1.0 %</td>
</tr>
</tbody>
</table>

4.4.3 Evaluation of 4-Channel DVR

Next, we validate our proposed technique by applying it to a practical example, 4-Channel digital video recorder (DVR). The raw bit streams from external 4 sources are encoded separately by DVR using an H.263 encoding algorithm. Figure 4-13(a) shows the specification of an H.263 encoding algorithm. All function blocks of an H.263 encoder except for the ME and the DCT blocks are mapped to one ARM720T. All ME and DCT blocks are mapped to the dedicated hardware blocks for ME and DCT respectively so that four H.263 encoders share two hardware blocks. Therefore the initial schedule of each function block is constructed like Figure 4-13(b). Table 4-3 reports the total memory access counts for each processing element. Memory traces are obtained by encoding a P-frame of QCIF-formatted bit-stream from the same video clip for all Channels.
Figure 4-13. (a) The specification of an H.263 encoding algorithm, (b) the initial schedule of 4-Channel DVR.

Table 4-3. The size of memory traces obtained from 4-Channel DVR.

<table>
<thead>
<tr>
<th>Processing element</th>
<th>Access counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM0</td>
<td>1,422,860</td>
</tr>
<tr>
<td>ARM1</td>
<td>1,422,860</td>
</tr>
<tr>
<td>ARM2</td>
<td>1,422,860</td>
</tr>
<tr>
<td>ARM3</td>
<td>1,422,860</td>
</tr>
<tr>
<td>ME</td>
<td>139,392</td>
</tr>
<tr>
<td>DCT</td>
<td>304,128</td>
</tr>
</tbody>
</table>

With the DVR example, we verified our proposed exploration flow that will be explained in Chapter 6. Many architecture candidates are generated changing the number of buses, the mapping of processing elements to buses, the move of related shared memory segments, and the associated bridge connections. Other
communication architecture parameters, such as priority assignment, bus data-width, and so on, are fixed to the arbitrary values.

The results of exploration are summarized in Table 4-4, Table 4-5, and Table 4-6. The number of buses is changed from 1 to 6. For a given number of buses, the number of bus bridges reflects indirectly the complexity of bus topology. In the last two columns on the right side, the estimation errors compared with the trace-driven simulation are reported with its range and the average absolute values. During the exploration over more than 200 architectures for each arbitration scheme, our proposed estimation technique keeps its accuracy within the range from –6% to 8.5%. The time for the trace-driven simulation is about 5 minutes for one bus architecture with fixed priority arbitration while the proposed scheme spends about 1.5 seconds as reported in Table 4-7. Through this experiment, we verified the validity of our proposed static estimation techniques.

Table 4-4. The experimental results about 4-Channel DVR for fixed priority.

<table>
<thead>
<tr>
<th>Number of buses</th>
<th>Number of explored architectures</th>
<th>Number of included bridges</th>
<th>Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0–0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-5.17~5.17</td>
</tr>
<tr>
<td>2</td>
<td>45</td>
<td>1–1</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-5.96~5.60</td>
</tr>
<tr>
<td>3</td>
<td>59</td>
<td>2–3</td>
<td>2.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-0.28~8.45</td>
</tr>
<tr>
<td>4</td>
<td>69</td>
<td>3–5</td>
<td>4.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-0.10~7.45</td>
</tr>
<tr>
<td>5</td>
<td>183</td>
<td>4–9</td>
<td>7.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-0.20~7.45</td>
</tr>
<tr>
<td>6</td>
<td>50</td>
<td>9–12</td>
<td>10.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-0.12~0.97</td>
</tr>
</tbody>
</table>

75
### Table 4-5. The experimental results about 4-Channel DVR for round-robin.

<table>
<thead>
<tr>
<th>Number of buses</th>
<th>Number of explored architectures</th>
<th>Number of included bridges</th>
<th>Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Range</td>
<td>Average</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0–0</td>
<td>0.0</td>
</tr>
<tr>
<td>2</td>
<td>45</td>
<td>1–1</td>
<td>1.0</td>
</tr>
<tr>
<td>3</td>
<td>59</td>
<td>2–3</td>
<td>2.7</td>
</tr>
<tr>
<td>4</td>
<td>69</td>
<td>3–5</td>
<td>4.7</td>
</tr>
<tr>
<td>5</td>
<td>181</td>
<td>4–9</td>
<td>7.6</td>
</tr>
<tr>
<td>6</td>
<td>50</td>
<td>9–12</td>
<td>10.8</td>
</tr>
</tbody>
</table>

### Table 4-6. The experimental results about 4-Channel DVR for two-level TDMA.

<table>
<thead>
<tr>
<th>Number of buses</th>
<th>Number of explored architectures</th>
<th>Number of included bridges</th>
<th>Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Range</td>
<td>Average</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0–0</td>
<td>0.0</td>
</tr>
<tr>
<td>2</td>
<td>45</td>
<td>1–1</td>
<td>1.0</td>
</tr>
<tr>
<td>3</td>
<td>59</td>
<td>2–3</td>
<td>2.7</td>
</tr>
<tr>
<td>4</td>
<td>114</td>
<td>3–6</td>
<td>4.9</td>
</tr>
<tr>
<td>5</td>
<td>71</td>
<td>4–9</td>
<td>7.5</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>11–11</td>
<td>11.0</td>
</tr>
</tbody>
</table>
4.4.4 Evaluation of On-Chip Bus Standard:

**AMBA AHB**

As a final experiment, we applied our estimation technique on the fixed priority based systems to one of the most popular on-chip bus standards, AMBA AHB [3], using the BFM, which consists of two modules, trace-fetcher and AHB-transaction-generator. Trace-fetcher reads burst transfers from the memory traces of a processing element. AHB-transaction-generator issues associated AHB transactions. Its operation is illustrated in Figure 4-14.

![Figure 4-14](image)

**Figure 4-14.** (a) Memory traces and (b) AHB transaction by memory accesses from the time 320 to the time 323 in (a): a *four-beat incrementing/wrapping burst transfer.*
**Chapter 4  Performance Estimation using Static Schedule**

![Image of graph showing estimation errors](image)

**Figure 4-15. Estimation errors of the proposed technique compared with the AHB bus using (a) the randomly generated memory traces and (b) 4-channel DVR.**

**Table 4-7. Estimation errors of the proposed technique compared with the AHB bus applied to 4-Channel DVR.**

<table>
<thead>
<tr>
<th></th>
<th>Estimation</th>
<th>BFM simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU time</td>
<td>1.5 sec</td>
<td>9571 sec</td>
</tr>
<tr>
<td>Execution cycles</td>
<td>29,729,922 cycles</td>
<td>29,961,733 cycles</td>
</tr>
<tr>
<td>Estimation error</td>
<td>-0.77%</td>
<td>-0.77 %</td>
</tr>
</tbody>
</table>

In our experimental environment, BFM can be instantiated as many as the number of processing elements used and all BFM s are attached to a single AHB bus with the asynchronous SRAM of 3-cycle access time. The entire AHB system
including BFM s and memory subsystems are written in VHDL of RTL level. Due to the implementation limit, a bus topology is restricted to the single bus and no retry/split transactions are allowed. We measured the error of the proposed technique for the fixed-priority based system compared with the simulation of the AHB system. The example schedule of Figure 4-10(a) and 4-Channel DVR are applied again to this experiment. As for the schedule in Figure 4-10(a), the numbers of processing elements are 2, 4, 6, 8, and 10. The memory traces of each processing element are generated randomly, as conducted in the first experiment. Table 4-7 shows the comparison results of our proposed technique and the simulation of the AHB system using BFM s. Although the more specific features of the AHB specification, such as handover after burst, default bus master granting, granting access with wait states, and so on, are not taken into account in our estimation, the estimation results show the validity of the proposed technique. As also confirmed in Table 4-7Figure 4-12(b), the speed-up of the proposed technique compared with the simulation using BFM s as well as its estimation accuracy makes our proposed technique viable for fast evaluation of the AHB system.

4.5 Chapter Summary

In this chapter, we presented an efficient static estimation technique that uses a static schedule of given application. In the proposed technique, the performance estimation based on the queuing model explained in Chapter 3 is applied to a set of
concurrently executable function blocks every beginning of function blocks according to the static schedule. Since the estimation time is pseudo-polynomial to the number of processing elements, the number of function blocks, and the number of buses used, the proposed technique can be used to prune the large design space before the trace-driven simulation. The proposed technique was also extended to consider multiple bus architectures. Experimental results show that our proposed technique is several orders of magnitude faster than the trace-driven simulation while keeping the estimation error within 8% consistently in the various communication architecture configurations.
Chapter 5

Performance Estimation

Considering Multitask Applications

5.1 Introduction

With the fast evolution of programmable hardware, such as microprocessor or DSP, and ever increasing complexity of its application, many parts of the applications tend to be implemented as software. Such systems let programmable hardware run multiple tasks, which brings up the need for new performance evaluation technique suitable to multitask applications. We propose the performance estimation technique being capable of considering multitask
applications, which is the extension of what are explained in Chapter 3 and Chapter 4.

The remainder of this chapter is organized as follows. Section 5.2 discusses the proposed estimation technique with a preliminary example. We show some experimental results in Section 5.3 and conclude this chapter in Section 5.4.

5.2 Estimation of Multitask Applications

This section discusses the extension of our previous performance estimation technique to multitask applications. For simplicity but with little loss of generality, we make the following assumptions: 1) All tasks are independent, 2) Any preemptable task scheduling policy can be used, and 3) The scheduling overhead is negligible.

We explain the proposed technique using a 4-Channel DVR. Each channel corresponds to a task so that DVR has four tasks, from ch0 to ch3. Figure 5-1(a) and Figure 5-1(b) show the specification and the schedule of an H.263 encoder respectively while Figure 5-1(c) represents a mapping example of function blocks onto processing elements. Each ARM processor takes charge of running two tasks respectively. A task is mapped to one ARM processor except for the function blocks ME and DCT, which are mapped to the dedicated hardware blocks, HE_ME and HW_DCT, respectively.
Figure 5-1. (a) The specification and (b) the schedule of H.263 encoder and (c) the mapping of function blocks to processing elements for 4-Channel DVR.
Suppose we want to estimate the execution time of \( ME0 \) of task \( ch0 \) mapped to \( HW\_ME \). To build the queuing model as explained in Section 4.1, the function blocks being concurrently executed on \( ARM0 \), \( ARM1 \), and \( HW\_DCT \) with \( ME0 \) should be selected. In \( ch0 \), no function blocks are concurrently executable with \( ME0 \) due to the execution dependency. Therefore, only \( ch1 \) can be simultaneously executable with \( ME0 \) in \( ARM0 \) since \( ME0 \) of task \( ch0 \) is executed in \( HW\_ME \). But, it cannot be statically determined which function block of \( ch1 \) is concurrently executable with \( ME0 \). Moreover, any function block of two tasks \( ch2 \) and \( ch3 \) can be executed in \( ARM1 \). If we enumerate all possible combinations of the function blocks concurrently executable with \( ME0 \), \( 5 \times \binom{2}{1} \times \binom{3}{2} \) or 150 queuing models should be investigated, which is impractical for fast design space exploration. Thus, we propose a heuristic approach.
Chapter 5  Performance Estimation Considering Multitask Applications

As explained in the beginning of this section, the required parameters for constructing the queuing model are the bus request rate and the average bus access time. To model the bus contentions due to other tasks, we define a virtual function block $VFB_{\tau}^{pe}$ for task $\tau$ on processing element $pe$, which has an approximated bus request rate $ar(VFB_{\tau}^{pe})$ and a bus access time $ba(VFB_{\tau}^{pe})$ while $MEO$ is executed. $ar(VFB_{\tau1,ARM0})$ and $ba(VFB_{\tau1,ARM0})$ of $VFB_{\tau1,ARM0}$ are computed as follows:

\[
ar(VFB_{ch1,ARM0}) = \sum \frac{bc(fb)}{sl(ch1)} \quad (5-1)
\]

and

\[
ba(VFB_{ch1,ARM0}) = \sum \frac{ba(fb_{ch1,ARM0}) \cdot bc(fb_{ch1,ARM0})}{bc(fb_{ch1,ARM0}}), \quad (5-2)
\]

where $fb \in \{the \ function \ blocks \ of \ ch1 \ mapped \ to \ ARM0\}$ and $sl(ch1)$ is the schedule length of the task $ch1$. In $ARM1$, two virtual function blocks, $VFB_{\tau2,ARM1}$ and $VFB_{\tau3,ARM1}$, are defined. Then, the virtual function block with higher bus request rate is selected. We also assume that the schedule length of all virtual function blocks is infinite to make them longer than $MEO$.

For more general formulation, we define two terms $T(fb)$ and $P(fb)$ that are the task including function block $fb$ and the processing element executing function block $fb$ respectively. If two function blocks $fb$ and $fb^*$ are to be executed concurrently according to the static task schedule, we denote it by $fb//fb^*$. Suppose that function block $fb^*$ is executed on processing element $pe^*$, i.e., $P(fb^*)=pe^*$. In
order to build the queuing system of Section 4.1, $\psi_{fb^*,pe}$ is defined as the virtual function block of processing element $pe$, which is assumed running concurrently with function block $fb^*$. Therefore, the bus request rate $ar(\psi_{fb^*,pe})$ of $\psi_{fb^*,pe}$ becomes

$$ar(\psi_{fb^*,pe}) = \max_{\forall \tau, \tau \neq *} \left\{ \frac{\sum bc(fb_i)}{sl(\tau)} \right\}, \quad (5-3)$$

where $fb_i \in \{ fb | T(fb) = \tau, P(fb) = pe, pe \neq pe^* \}$ and $\tau^* = T(fb^*)$. If task $\tau$ is selected to build the virtual function block of $pe$ by Equation (5-3), the average bus access time $ba(\psi_{fb^*,pe})$ of virtual function block $\psi_{fb^*,pe}$ in ideal bus conditions is

$$ba(\psi_{fb^*,pe}) = \frac{\sum \{ ba(fb_i) \cdot bc(fb_i) \}}{\sum bc(fb_i)}, \quad (5-4)$$

where $fb_i \in \{ fb | T(fb) = \tau, P(fb) = pe, pe \neq pe^* \}$. With those queuing parameters of the processing element, the queuing system is constructed to estimate the average wait time for bus grant of function block $fb^*$.

### 5.3 Experiments

We compared the static estimation result from the proposed multitask extension of the previous queuing analysis for the 4-Channel DVR system example with a trace-driven simulation. Trace-driven simulation used in this experiment adopts a
simple bus protocol, where the advanced features such as address/data bus pipelining, split-transaction, multiple-outstanding masters, and so on are not modeled. The trace-driven simulator schedules tasks using the rate-monotonic scheduling with fixed priorities [33]. However no scheduling overhead is considered in the simulator.

Table 5-1. The results of exploration for 4-Channel DVR.

<table>
<thead>
<tr>
<th>Number of buses</th>
<th>Number of architectures</th>
<th>Estimated execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>24869021</td>
</tr>
<tr>
<td>2</td>
<td>2208</td>
<td>22297455</td>
</tr>
<tr>
<td>3</td>
<td>1072</td>
<td>22502381</td>
</tr>
<tr>
<td>4</td>
<td>512</td>
<td>22486609</td>
</tr>
</tbody>
</table>

Total 3793

Estimation/arch in 1st step (sec) 0.83

CPU time (sec) 3918

Table 5-1 represents the results after architecture exploration for 4-Channel DVR until no more performance gain is obtained. In Table 5-1, the second column ‘Number of architectures’ shows the number of generated architecture candidates according to the associated number of buses during the exploration. The second row from the bottom indicates the average elapse of estimating an architecture candidate in the first exploration step. It takes no less than one second, which shows the effectiveness of the proposed technique. In the third column ‘Estimated
Chapter 5  Performance Estimation Considering Multitask Applications

execution time’, the best performance obtained from the trace-driven simulation is recorded in bus clock cycles. No performance gain is obtained with more than 3 buses, since the overhead of crossing bus bridges tends to exceed the benefit of scattering communication traffics by splitting a bus.

Figure 5-3 shows the average bus access time of unit data from each processing element on 4-Channel DVR by both the proposed static estimation and trace-driven simulation results over the explored architectures. The estimation error compared
with the trace-driven simulation is about 28% in the worst case. Since the ratio of bus access time over the entire execution does not exceed 30%, the estimated error on the entire execution becomes less than 10%. Furthermore, the average estimation error is around about 6%. Through the experiments, we verify that the proposed estimation technique can be used successfully for reducing the design space for multitask applications.

5.4 Chapter Summary

In this chapter, we proposed the static performance estimation technique of on-chip bus architectures for multitask applications was proposed. The proposed technique is based on the queuing model of processing elements associated with a set of function blocks that can be executed concurrently. To construct a set of function blocks considering concurrency between tasks, a virtual function block corresponding to a task in each processing element is introduced with approximated parameters for the queuing analysis. Experimental results show that the proposed technique is acceptable for the fast design space exploration of communication architecture with reasonable estimation error.
Chapter 6

Exploration Framework

6.1 Introduction

Separation between computation and communication enables the system designer to explore the communication architecture independently of component (or processing element) selection and mapping. In this paradigm, communication architecture decision is performed after a decision is made on which processing elements are used and which function blocks are mapped to where. From the given communication requirements from all processing elements, the design space of communication architectures is explored to determine the optimal one considering the trade-offs between performance, power, cost, and other design objectives. As
we discussed previously, it is critical to develop an efficient exploration technique because the design space of communication architectures is extremely wide.

This chapter proposes a framework that incorporates where all steps are incorporated into, which includes performance estimation technique discussed through Chapter 3, Chapter 4, and Chapter 5. In the framework, first, we use a static performance estimation technique to quickly evaluate each candidate design point and prune the design space drastically. Then, in the next step, trace-driven simulation is used to accurately evaluate the design points in the reduced space and determine a set of pareto-optimal set of bus architectures.

We assume that the processing elements communicate with each other through a shared memory. Each processing element has a single port for both local and shared memory accesses, as usually is the case in real systems. Then, local memory traffics as well as shared memory traffics are also involved in bus contention: Memory allocation is considered as an important axis of the design space in our technique. On the other hand, most previous works [25][26] have only considered shared memory traffic so that they do not consider memory allocation separately.

The remainder of this chapter is organized as follows. Section 6.2 reviews some related works. The details of the proposed technique will be provided with a preliminary example through Section 6.3 and 6.4. Section 6.5 provides the overall structure of the proposed exploration framework. We show some experimental results in Section 6.6 and conclude this chapter in Section 6.7.
6.2 Related Work

Some researchers have considered communication architecture selection simultaneously during the mapping step. Since the communication overhead is needed for the mapping decision, static estimation of communication architecture has been investigated. A technique was proposed to estimate the communication delay using the worst-case response analysis of real-time scheduling [6]. Knudsen and Madsen estimated the communication overhead on a point-to-point channel taking into account the data transfer rate variation depending on the protocol, configuration, and different operating frequencies of components [5]. Nandi et al. proposed a performance measure technique based on continuous-time Markov processes [27]. However, these techniques did not model the dynamic effects such as bus contention and explored only a limited configuration space.

For exploration of communication architectures, simulation-based estimation is widely adopted in many commercial tools and academic researches at various transaction levels [12][13][15]. A simulation-based method gives accurate estimation results but pays too heavy computational cost to be used for exploring the large design space. So, researches based on this method cannot but exploit only a few design axes to reduce the design space significantly. To overcome this difficulty, a hybrid approach between a static estimation and a simulation approach has been developed by Lahiri et al. [19]. They used some static analysis to group the traces and apply a trace-driven simulation with the trace groups. Their approach
is similar to ours in that they applied some static analysis to the memory traces to reduce the time complexity of trace-driven simulation.

Since the design space is extremely huge, most previous works focused on a small number of design axes. In Gong et al.’s work [28], system specification refinement onto four fixed communication architecture templates was addressed to optimize performance. Gasteier et al. proposed a bus topology synthesis technique at high level using the port constraints of components and considering only static information such as bit widths, the amount of data transfer, and so on [29]. Meeuwen et al. presented a technique for cost-efficient interconnect architecture exploration by time-multiplexing the data transfers over a number of shared buses assuming distributed memory systems under pre-determined memory allocation [30]. Meftali et al. found performance-optimal shared memory allocation considering area of communication channel and memory subsystem using ILP in a point-to-point communication architecture [31]. Lahiri et al. proposed an exploration technique optimizing the component mapping to bus and the bus protocol such as DMA block transfer size and bus priority assignment for a given bus topology [25]. Srinivasan and Vijaykrishnan developed a technique that performs both bus partitioning and bus frequency assignment simultaneously to optimize power consumption and performance using a genetic algorithm [58]. A technique using the profiled statistics of communication traffics between cores to determine core-to-bus assignment for a given application was proposed by Drinic et al. [10].
6.3 Generation of Communication Architecture Candidates

In this section, we explain how the proposed technique explores the design space of 4-Channel DVR. Suppose that each H.263 encoder is mapped to a separate ARM processor except all ME and DCT blocks, which are mapped to a ME hardware component and a DCT hardware component respectively. Thus, four H.263 encoders share two hardware components.

Figure 6-1. (a) The initial schedule of 4-Channel DVR and (b) its single-bus implementation.
Chapter 6 Exploration Framework

Figure 6-1(a) and Figure 6-1 (b) show the scheduling and mapping result of 4-Channel DVR and its single-bus implementation respectively. This example system has 14 memory segments: 5 local memory segments and 12 shared memory segments. In Figure 6-1(b), for instance, shared memory segment ‘MC0, ME0’ is associated with communication between function blocks MC0 and ME0.

6.3.1 Bus Topology Exploration

Performance improvement of communication architecture can be done by scattering communication traffics into multiple buses to reduce the contentions and to maximize concurrency. For this purpose, we select a processing element and allocate it to a new bus or to another existing bus. Suppose that we select ARM0 in the single-bus architecture of Figure 6-1. The way of changing the allocation of ARM0 is only to create a new bus, Bus1. Then, its associated shared memory segments ‘MC0, ME0’ and ‘DCT0, Q0’ can be allocated to either Bus0 or Bus1 to generate four architecture candidates as shown in Figure 6-2. It is important to note that a bus bridge is introduced between two buses for inter-bus communication.
Figure 6-2. Creating a new bus for processing element \textit{ARM0} and allocating its associated shared memory segments.

Figure 6-3. If the local memory segment of processor \textit{ARM0} is separated from shared bus \textit{Bus1}, communication traffics incurred by local memory accesses can be removed from \textit{Bus1}, which may lead to further performance improvement.
Further communication traffics reduction can be obtained by removing local memory accesses of each processing element from shared buses as illustrated in Figure 6-3. Processor ARM0 can access its local memory segment LM_ARM0 without contending for the use of Bus1. A drawback of such separation is area overhead caused by adding a local bus to each processing element. It, however, may be desirable for power saving due to the use of more memory components of smaller size as well as reduced bus-switching activities [59][60]. We leave it as a future work to consider power consumption as another design objective.

Table 6-1. The number of generated architectures from single-bus architecture of 4-Channel DVR by changing allocation of shared memory segments.

<table>
<thead>
<tr>
<th>PE being moved</th>
<th>ARM0</th>
<th>ARM1</th>
<th>ARM2</th>
<th>ARM3</th>
<th>ME</th>
<th>DCT</th>
</tr>
</thead>
<tbody>
<tr>
<td># of shared memories</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td># of generated architectures</td>
<td>$2^2$</td>
<td>$2^2$</td>
<td>$2^2$</td>
<td>$2^2$</td>
<td>$2^8$</td>
<td>$2^8$</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>528</td>
<td></td>
</tr>
</tbody>
</table>

Table 6-1 shows the size of design space by moving each processing element from the single-bus architecture in Figure 6-1(b). 528 architecture candidates are generated in total. The same number of architecture candidates is also generated for the architectures that use local buses for local memory segments. Consequently,
more than a thousand architecture candidates should be investigated by examining the bus topology only.

### 6.3.2 Bus Protocol Synthesis

Once the bus topology and memory allocation are determined, bus protocol of each architecture candidate should be configured. Bus protocol includes priority assignment, operation clock frequency, bus data-width, and so on. Of these, priority assignment should be treated with care since it may cause significant performance variation as observed in [25] and [56]. Although an exhaustive search method guarantees an optimal result, it is prohibitively expensive. For example, the first architecture, \( \text{arch0} \), of Figure 6-2 have 6 bus masters in \( \text{Bus0} \) and 2 bus masters in \( \text{Bus1} \) including bus bridges. Since 6! or 1440 priority assignments for \( \text{Bus0} \) and 2! assignments for \( \text{Bus1} \) are possible, 1440×2 assignments should be investigated in an exhaustive search method to get the optimal priority assignment. To overcome this difficulty, we devised a priority assignment heuristic where a higher priority is bestowed to the processing element with more memory accesses and to more critical processing element.

The amount of data transfer per unit time \( BW(fb_i) \), \( i.e., \) bandwidth, represents the memory access characteristics of function block \( fb_i \). The criticality \( C(fb_i) \) of function block \( fb_i \) is the sum of the schedule length of function blocks on the longest execution path starting from block \( fb_i \). The bandwidth and the criticality of
a function block are computed from the memory traces and the function block scheduling respectively. In our heuristic, we define the rank $R(f_{bi})$ of function block $f_{bi}$ as the product of criticality $C(f_{bi})$ and bandwidth $BW(f_{bi})$. Also the rank $R(PE)$ of processing element $PE$ is the sum of the ranks of function blocks that are executed in $PE$. Thus, we get following formula:

$$R(PE) = \sum_{f_{bi} \in FB_{PE}} R(f_{bi}) = \sum_{f_{bi} \in FB_{PE}} [BW(f_{bi}) \cdot C(f_{bi})],$$

(6-1)

where $FB_{PE}$ is the set of function blocks mapped on $PE$. The higher the rank of a processing element is, the higher priority the processing element is assigned. After this initial assignment, we perform a simple annealing process by swapping the priorities of two processing elements on the same bus.

When the priority assignment of a bus is investigated, assignments of the other buses are assumed fixed. For instance, in arch0 of Figure 6-2, we start by varying the priorities of processing elements in Bus0 first. The performance estimation proposed in the previous section is applied to every combination by swapped priorities between two processing elements. Therefore, $\binom{6}{2}$ or 30 architecture candidates are investigated when considering Bus0 and the best result is selected as the priority assignment of Bus0. Then we move to Bus1 and repeat the same procedure to find a more improved solution, which results in $\binom{3}{2}$ or 6 priority assignments. Therefore, the total number of assignments to be explored by the
heuristic amounts to 36. The proposed assignment heuristic shows remarkable results compared with an exhaustive search method, while it explores significantly reduced design space, by 80 times in this example. We validate its efficiency by experimental results in Section 6.6.

Bus clock frequency and bus data-width depend on the memory used. For brevity, in this paper, we assume that all buses in an architecture candidate are synchronized with a single global clock and its frequency is set to be the reciprocal of memory access time for one word. Bus data-width follows the data-width of memory.

### 6.4 Trace-Driven Simulation

After reducing a large design space by the proposed static performance estimation using the queuing analysis, architecture candidates in the reduced design space are evaluated using cycle-accurate simulation to choose the best candidate to use it as a seed in the next iteration. It should be noticed that the purpose of this step is an accurate evaluation of each qualified architecture candidates rather than fast exploration since the most of an initial search space were already pruned in the first exploration step by the static performance estimation. This section describes an implementation of cycle-accurate trace-driven simulation used in the second exploration step.
Contrary to execution-driven simulation, trace-driven simulation uses traces of memory accesses from processing elements to be simulated. In our framework, memory traces are collected by initial run of co-simulation using ISS for microprocessors and HDL simulator for dedicated hardware blocks or legacy IPs. When obtaining memory traces, we do not take into account any kind of overhead by communication architectures, i.e. no contentions on bus and one cycle for unit word access. Such ideal but unrealistic memory accesses are lengthened according to target bus architecture involved in simulation.

Figure 6-4. The overall structure of trace-driven simulator.

We perform trace-driven simulation with the following three inputs: a target architecture description, memory traces from initial co-simulation, and the
Chapter 6 Exploration Framework

schedule of function blocks in an application. Figure 6-4 shows the overall structure of the trace-driven simulator. It mainly consists of three parts, which are processing elements, networks of buses, and memory subsystems.

6.4.1 Component Model

**Processing Element**

A processing element is modeled with two states: *INT_EXE* and *MEM_ACC*.

- *INT_EXE*: This state indicates that no bus request or no bus access is represented by a processing element. This state includes behaviors of a processing element such as being idle, cache hit, and internal computation. If we consider the trace example in Figure 4-14(a), the period between time stamp 310 and 320 corresponds to this state. In the trace-driven simulation, it is not important no matter what the period is actually. At the beginning of this state, for example, at the very next cycle of time stamp 310, the simulator reads the next burst transfer group from a trace file, which starts at time stamp 320. Then, it waits for 10 bus cycles to model an internal behavior of the processing element and issues a bus request changing a current state to *MEM_ACC*.

- *MEM_ACC*: The wait time for a bus grant as well as actual memory accesses is included in this state. Once the network of buses detects requests accumulated, a bus grant is made on each bus according to arbitration policy and
an associated processing element is informed. Then, the granted processing element starts to access a memory in burst transfer operation.

**Memory Subsystem**

Memory subsystems are modeled simply with two parameters, `burstInit` and `latency`. `burstInit` corresponds to an initialization time to start a burst transfer. `latency` is an access time of an individual sequential access in a burst transfer. If a processing element wants to access eight unit words in burst from a memory subsystem with `burstInit` set to 2 and `latency` set to 1, it would take $2 + 8 \times 1 = 10$ cycles to complete the burst transfer. Note that a global single clock is fed to an entire bus network.

**Network of Buses**

We model a bus using the following five states:

- **IDLE**: In this state, no bus request is being served. A bus is waiting for a request from processing elements. Bus arbitration occurs every bus cycle in this state for accumulated bus requests *if* any.

- **ADDR_DRIVE**: A bus judges where a granted bus request is heading for by decoding the address of the granted request. If a targeted memory is on other bus (destination bus), an associated bus bridge initiates a new request on the destination bus as a processing element. Then, the current bus (source bus) enters the state...
BRIDGE_WAIT waiting for a bus grant from the destination bus. Otherwise, the next state would be MEM_INIT to start a memory access on the current bus.

- **BRIDGE_WAIT**: A bus in this state is waiting until a bus request initiated from the source bus is granted in the destination bus. When a processing elements in bus A wants to access bus B and another in bus B also wants to access bus A at the same time, a deadlock may occur. If a deadlock is detected, one of the bus requests should be withdrawn, which is the newer one of them in current implementation.

- **MEM_INIT**: This state is associated with the initialization phase of a burst transfer.

- **MEM_TRANSFER**: Actual memory transfer is made in this state, which consists of consecutive accesses of unit word. Furthermore, if any bus request arrived before current transfer is finished, bus arbitration should be made during a last data transfer at the same time to model a pipelined on-chip bus protocol.

### 6.4.2 Putting It All Together

Figure 6-5 shows the pseudo code of trace-driven simulator. The outermost loop corresponds to the jobs of all processing elements and a bus network during a bus clock period. For each bus involved in a network of buses, the jobs for the processing elements connected to current bus (line 3 to line 5) and the jobs for buses according to their own state are described in this loop (line 6 to line 34). In
the procedure `peStateMachine` of line 4, associated jobs are processed regarding their current state. The details of jobs to be done in each state were explained in the previous section. For the brevity of explanation, we omit the parts where an execution sequence of processing elements is determined according to a given schedule of function blocks.

```
while (true) {
    for each bus {
        for each PE {
            peStateMachine();
        }
        switch according to currentBusState {
            case BUS_IDLE:
                if (bus request is presented)
                    nextBusState=BUS_ADDR_DRIVE;
                case BUS_ADDR_DRIVE:
                    getCurrentBusMaster();
                    translatedAddr = getTranslatedAddr();
                    if (translatedAddr is for the current bus)
                        nextBusState=BUS_MEM_INIT;
                    else if (translatedAddr is for other buses)
                        nextBusState=BUS_BRIDGE_WAIT;
                    else
                        error();
            case BUS_MEM_INIT:
                getCurrentBusMaster();
                if (initializationCount is zero)
                    nextBusState=BUS_MEM_TRNSFER;
                else
                    initializationCount--;
            case BUS_MEM_TRNSFER:
                getCurrentBusMaster();
                if (burstCount==0) {
                    if (a bus request is presented)
                        nextBusState=BUS_ADDR_DRIVE;
                    else
                        nextBusState=BUS_IDLE;
                } else burstCount--;
        } // end of 'switch according to currentBusState'
    } // end of 'for each bus'
    updateBusStatus();
    curTime++; // advance time by 1 cycle;
    if (all tasks are completed) break;
} // end of 'while'
```

Figure 6-5. The pseudo code describing the trace-driven simulation.

### 6.5 Exploration Algorithm

Figure 6-6 summarizes the main procedure of the proposed technique:

**Select_Architecture.** This procedure requires three inputs: the initial architecture
Chapter 6  Exploration Framework

Initial_Arch to begin the exploration, the schedule information of system specification Sched, and the memory traces Mem_Trace of processing elements. The while statement of line 3 defines the main iteration loop of exploration.

Select_Architecture consists of three parts. The first part is the first architecture-pruning step from line 5. Initially, the set of architecture candidate contains only one element, Initial_Arch. In the first for loop, from line 6 to line 9, the diverse priority assignments selected from the proposed priority assignment heuristic are assessed by the proposed static estimation method and we obtain the best performance of each architecture candidate.

Then, after the best performance values of all architecture candidates are sorted in an ascending order, the architecture that has the shortest execution time, Best_Exe_Time, is chosen. Since our static estimation method is observed to have 10% error bound through preliminary examples, the architecture candidates that have the estimated performance differed from Best_Exe_Time by less than 10% may have actually better performance than Best_Exe_Time. Thus, this error range is used to reduce the design space: The parameter ESTIMATION_ERROR is set to 0.1. In the for loop from line 11 to 18, if the performance difference of an architecture candidate from Best_Exe_Time is greater than ESTIMATION_ERROR, it is pruned from the design space. Note that the more accurate the static estimation technique is, the narrower becomes the design space that should be investigated more precisely in the second pruning step.
In the case the reduced design space is still too large, we may want to restrict the maximum number of architecture candidates to be explored in the second step. Therefore, we define the \textit{MAX\_ARCH} parameter and enforce that at most as many as \textit{MAX\_ARCH} architecture candidates are left in the reduced design space (line 19 to line 25).

The second part of the procedure applies trace-driven simulation to the selected architecture candidates from the first step. We use in-house cycle-accurate trace-driven simulator at this step. Since the estimated performance from trace-driven simulation is very accurate, we compare the performances of all candidate architectures and choose the best architecture. Then the performance of the best architecture is compared with that of the previous iteration. If no performance improvement is achieved from the current iteration or the number of shared buses reaches the number of processing elements, we exit the iteration loop and terminate the procedure.

The last part of procedure \textbf{Select\_Architecture} is to generate the architecture candidate incrementally from the best architecture chosen from the second part (line 39). How to generate the architecture candidate is already explained in the previous section. When we estimate the performance, we record the best performance value for each number of buses used to obtain the pareto-optimal set of bus architectures.
**Chapter 6  Exploration Framework**

```python
1: Select_Architecture (Initial_Arch, Sched, Mem_Trace)
2:   arch_list Initialize(Initial_Arch)
3:   while (true) do
4:     Num_Qualified_1st = 0
5:     // 1st pruning step
6:     for each arch \(i \in \text{arch_list}, i=1,2,...,N_{1st}\) do
7:       Bus_Protocol_Synthesis(arch)
8:       Do_Performance_Estimation(arch, Sched, Mem_Trace)
9:     end for
10:    Best_Exe_Time = Find_Best_Exe_Time(arch_list)
11:   for each arch \(i \in \text{arch_list}, i=1,2,...,N_{1st}\) do
12:     if ((arch \(i\) \(\rightarrow\) Exe_Time - Best_Exe_Time) / Best_Exe_Time > ESTIMATION_ERROR) then
13:       arch_list \(\rightarrow\) Delete(arch)
14:     else
15:       Num_Qualified_1st = Num_Qualified_1st + 1
16:     end if
17:   end for
18:   end if
19:   if (Num_Qualified_1st > MAX_ARCH) then
20:     for each arch \(i \in \text{arch_list}, i=1,2,...,N_{1st}\) do
21:       if (i > Num_Qualified_1st) then
22:         arch_list \(\rightarrow\) Delete(arch)
23:       end if
24:     end for
25:   end if
26: else  // 2nd pruning step
27:     Prev_Seed_Arch = Curr_Seed_Arch
28:   for each arch \(i \in \text{arch_list}, i=1,2,...,N_{2nd}\) do
29:     Do_Trace_Driven_Simulation(arch, Sched, Mem_Trace)
30:   if (Curr_Seed_Arch \(\rightarrow\) Exe_Time < arch \(i\) \(\rightarrow\) Exe_Time) then
31:     Curr_Seed_Arch = arch \(i\)
32:   end if
33: end for
34: // check the termination condition
35: if ((Curr_Seed_Arch \(\rightarrow\) Exe_Time >= Prev_Seed_Arch \(\rightarrow\) Exe_Time) or
36:   (Curr_Seed_Arch \(\rightarrow\) Num_Shared_Bus == Curr_Seed_Arch \(\rightarrow\) Num_Pe)) then
37:   Quit_Exploration()
38: end if
39: Generate_Architecture_Candidates(arch_list, Curr_Seed_Arch)
40: end while
41: end Select_Architecture
```

**Figure 6-6. The proposed exploration flow.**
6.6 Experiments

In this section, we validate our proposed exploration method by some experimental results. First, we compared the efficiency of the proposed priority assignment heuristic with the exhaustive assignment for architecture candidates during exploration of two examples; the equalizer subsystem of an OFDM DVB-T receiver and a 4-Channel DVR system, starting from the single-bus architecture. In a DVB-T receiver, the equalizer is used for correcting the amplitude distortion of received signals [32]. Function blocks of the equalizer are mapped onto 5 ARM940T processors and are scheduled in a pipelined fashion to make all processors run concurrently as shown in Figure 6-7(b).

![Diagram](image)

**Figure 6-7.** (a) The specification of the equalizer subsystem for OFDM DVB-T receiver and (b) its schedule.
Due to excessively long run time of exhaustive search, we only considered single-bus and dual-bus implementations for performance comparison. Comparison results are summarized in Table 6-2 and Table 6-3 for each example respectively.

For each bus topology, the number of investigated combinations of priority assignment by exhaustive search and the heuristic are given in the rows ‘# of total assignments per architecture’ and ‘# of average assignments per architectures’ in their total and average respectively. Comparing with the exhaustive search, the proposed heuristic assignment reduces the search space significantly by about 12 times in a single-bus implementation and about 19 times in dual-bus implementations. We evaluated the performance of all combinations of priority assignment by exhaustive method and then recorded the normalized values with respect to the best whose performance is set to 1. The smaller the value is, the better the performance is. It should be noticed that wrong priority assignment leads to 30% performance degradation in the worst case. It confirms the importance of optimal priority assignment.

The column ‘Initial / Tuning’ reports the results by the proposed heuristic. The first value is obtained from the initial assignment while the second one is after the annealing process. As shown in the tables, initial assignment does not always guarantee acceptable results. Even though the quality of initial assignment only is not good, it can be elevated by the annealing process almost close to the optimum. In both examples, the heuristic results are deviated just different from the optimum only by 1% at most for various bus architectures. In the case of singe-bus
implementation of 4-Channel DVR, the optimum was found by the heuristic. It shows that the proposed heuristic is effective to find an optimized priority assignment as well as to reduce the search space drastically.

Table 6-2. Efficiency of the priority assignment heuristic compared with an exhaustive method: The equalizer subsystem for OFDM DVB-T receiver.

<table>
<thead>
<tr>
<th>Target architecture</th>
<th>Single bus</th>
<th>Dual Bus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assignment method</td>
<td></td>
<td></td>
</tr>
<tr>
<td># of architectures</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td># of total assignments per architecture</td>
<td>120</td>
<td>10</td>
</tr>
<tr>
<td># of average assignments per architecture</td>
<td>120</td>
<td>10</td>
</tr>
<tr>
<td>Performance</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Variation/Initial / Tuning</td>
<td>1 ~ 1.32</td>
<td>1.18 / 1.006</td>
</tr>
</tbody>
</table>

Table 6-3. Efficiency of the priority assignment heuristic compared with an exhaustive method: 4-Channel DVR.

<table>
<thead>
<tr>
<th>Target architecture</th>
<th>Single bus</th>
<th>Dual Bus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assignment method</td>
<td></td>
<td></td>
</tr>
<tr>
<td># of architectures</td>
<td>1</td>
<td>183</td>
</tr>
<tr>
<td># of total assignments per architecture</td>
<td>720</td>
<td>15</td>
</tr>
<tr>
<td># of average assignments per architecture</td>
<td>720</td>
<td>15</td>
</tr>
<tr>
<td>Performance</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Variation/Initial / Tuning</td>
<td>1 ~ 1.22</td>
<td>1.19 / 1.000</td>
</tr>
</tbody>
</table>
Chapter 6  Exploration Framework

In the second set of experiments, the proposed two-phase exploration technique is applied to the previous examples and additional 3 example systems. Figure 6-8 displays the specification of these 3 example systems. Edges between function blocks assigned to different processing elements denote shared memories. The maximum number of architectures to be evaluated in the second pruning step, $MAX\_ARCH$ in Figure 6-6, was fixed to 20. Figure 6-4 represents the results of exploration for each example system.

![Figure 6-8](image_url)

Figure 6-8. Three example systems: (a) System1, (b) System2, and (c) System3.
Table 6-4 represents the results of exploration for each example system. In each set of Table 6-4, the first column ‘\# of arch’ shows the number of generated architecture candidates having the associated number of buses during whole exploration. The number of processing elements of a system is the maximum number of buses that an architecture candidate can have.

The column ‘Speed up’ shows the performance improvement of the best architecture among the architecture candidates compared with the initial single-bus architecture. Performance improvement tends to be saturated near the end of exploration. It is noteworthy that the maximum performance of example systems is about 50% to 100% better than that of the single bus architecture. Since such improvement comes from optimization of only communication architecture and memory allocation, it confirms the usefulness of the proposed technique.

The four rows from the bottom represent the number of total architecture candidates explored, the architecture pruning ratio by the static performance estimation, the average time taken for the static performance estimation of an architecture candidate, and the total elapsed time for the exploration respectively. In the case of 4-Channel DVR, a pruning ratio is close to 100%. The entire set of architecture candidates includes the ones by priority assignments as well as by the move of processing elements and shared memories. As reported in the second row from the bottom, each architecture candidate is evaluated rapidly within less than one second in average. The total execution time of the last row includes trace-driven simulation.
Table 6-4. The experimental results of exploration for 5 example systems.

<table>
<thead>
<tr>
<th>Application</th>
<th>System1</th>
<th>System2</th>
<th>System3</th>
</tr>
</thead>
<tbody>
<tr>
<td># of shared mem.</td>
<td>3</td>
<td>14</td>
<td>19</td>
</tr>
<tr>
<td># of buses</td>
<td># of arch</td>
<td>Speed up</td>
<td># of arch</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>119</td>
<td>1.4443</td>
<td>4284</td>
</tr>
<tr>
<td>3</td>
<td>73</td>
<td>1.5592</td>
<td>2537</td>
</tr>
<tr>
<td>4</td>
<td>46</td>
<td>1.6387</td>
<td>1444</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>1.6390</td>
<td>242</td>
</tr>
<tr>
<td>6</td>
<td>-</td>
<td>-</td>
<td>48</td>
</tr>
<tr>
<td>7</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Total</td>
<td>250</td>
<td>8565</td>
<td>43712</td>
</tr>
<tr>
<td>Pruning ratio (%)</td>
<td>93.6</td>
<td>89.1</td>
<td>95.7</td>
</tr>
<tr>
<td>Estimation time in 1st step (sec)</td>
<td>0.07</td>
<td>0.36</td>
<td>0.82</td>
</tr>
<tr>
<td>Total exe. (sec)</td>
<td>17</td>
<td>3239</td>
<td>36234</td>
</tr>
</tbody>
</table>
Figure 6-9. Performance variations of 5 example systems during the exploration according to the number of buses. In each graph, horizontal and vertical axes represent the number of buses used and the normalized execution time respectively.

The performance variation of each system according to the number of buses is shown in Figure 6-9. For all systems, dual-bus system has the widest performance variation meaning that wrong mapping of processing elements or memory allocation could lead to significant performance degradation. For example, in 4-
Chapter 6  Exploration Framework

Channel DVR and System3, the worst performance of dual-bus architecture is even inferior to single-bus architecture. However, as more buses are used, the variation becomes smaller since the concurrency of memory accesses is fairly exploited by multiple buses enough to compensate performance degradation due to wrong mapping of processing elements and memory allocation. If we take the best performance for each number of buses, we obtain the pareto-optimal set of bus architectures.

As the last experiment, we examined how much performance improvement can be obtained by using dedicated local buses for local memory access as discussed in Section 6.3.1. The same environment and configurations of the previous experiment are used again for equalizer and DVR examples. Contrary to the previous experiment, local buses of processing elements are explored to get further performance improvement. Similar tendencies could be observed as shown in Table 6-4 and Figure 6-9. Now, we focus on how much performance improvement is obtained and summarize the results in Table 6-5.

**Table 6-5. Performance improvement obtained by considering local bus exploration.**

<table>
<thead>
<tr>
<th>Applications</th>
<th>Equalizer for OFDM DVB-T Receiver</th>
<th>4-Channel DVR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial single bus</td>
<td>1.00 (96,478 cycles)</td>
<td>1.00 (21,372,362 cycles)</td>
</tr>
<tr>
<td>Not considering local bus</td>
<td>0.67 (64,704 cycles)</td>
<td>0.57 (12,086,437 cycles)</td>
</tr>
<tr>
<td>Considering local bus</td>
<td>0.57 (55,012 cycles)</td>
<td>0.48 (10,227,737 cycles)</td>
</tr>
</tbody>
</table>
Chapter 6  Exploration Framework

For each example, performance values for three types of architecture are reported: Initial single bus architecture, the best architecture without local buses, and the best one with the local buses. Performance values are provided in both normalized ones and bus clock cycles. The performance of initial single bus implementation is set to 1. The performance improvement with local buses is about 100%, i.e., it becomes 2 times faster than initial single bus architectures for both examples. It is about 20% better than the architecture without local buses.

The final bus architectures obtained by the proposed exploration are shown in Figure 6-10 for equalizer and Figure 6-11 for 4-Channel DVR respectively. For both examples, in spite of superior performance, the number of shared buses tends to become smaller when local buses are considered during exploration. It implies that a large portion of communication traffics is due to local memory accesses.
Figure 6-10. The final bus architectures of the equalizer subsystem for OFDM DVB-T receiver: (a) without considering local bus and (b) with considering local bus.
Figure 6-11. The final bus architectures of equalizer for 4-Channel DVR: (a) without considering local bus and (b) with considering local bus.
6.7 Chapter Summary

In this chapter, we presented a framework for the iterative two-step exploration technique for on-chip bus architectures and memory allocations. At each of iteration, the first step reduces the large design space drastically and quickly by using an efficient static performance estimation method based on queuing model. In the second step, the reduced design space is explored using a trace-driven simulator to choose the best architecture candidate. Experimental results with preliminary examples, 4-Channel DVR and the equalizer subsystem for OFDM DVB-T receiver, and three randomly generated examples validate the efficiency and the viability of the proposed technique to explore the wide design space. The optimized bus architectures were almost 2 times faster than initial ones. Through the experiments, it has been proven that significant performance improvement can be made with bus architecture optimization only, which validates the viability of the proposed design methodology in this thesis.
Chapter 7  Conclusions and Future Directions

In this thesis, we proposed the methodology for customizing the design of the communication architecture to exploit the characteristics of the on-chip communication traffic generated by application, in particular, being concerned about bus-based communication architectures. Since the design space of communication architectures that we should consider is extremely wide, in the proposed technique, we utilized the advantages of both approaches by breaking down the exploration procedure into two steps.

To consider extremely wide design space, for the first step first, we presented an efficient static estimation technique of bus architectures based on three commonly used arbitration schemes: fixed priority, round-robin, and two-level TDMA, which
Chapter 7  Conclusions and Future Directions

is based on the queuing model and makes use of the schedule information and the memory traces. Then we presented an efficient static estimation technique that uses a static schedule of given application. Experimental results show that our proposed technique is several orders of magnitude faster than the trace-driven simulation while keeping the accurate estimation in the various communication architecture configurations.

Then an iterative two-step exploration technique was presented to optimize performance of bus-based on-chip communication architecture and memory allocation. At each of iteration, the first step reduces the large design space drastically and quickly by using an efficient static performance estimation method based on queuing model as mentioned above. In the second step, the reduced design space is explored using a trace-driven simulator to choose the best architecture candidate. Experimental results with some preliminary examples including DVR and the equalizer subsystem for OFDM DVB-T receiver validated the efficiency and the viability of the proposed technique to explore the wide design space.

As the future directions, Finally, we address the followings as the future directions: The bus architectures we assume in Chapter 4 do not have the advanced features being used in the-state-of-art bus architectures such as split-transaction, multiple outstanding bus masters, out-of-order transaction, and so on. Such advanced features will affect the service rate of the memory system in our simplified queuing model. It will be a future research topic to investigate the scope
Chapter 7 Conclusions and Future Directions

of applicability of the proposed technique. We believe that some features can be modeled easily as the overhead due to the pipelined operation of the AHB specification is considered in Section 4.4.4. Since the objective of the proposed static analysis is to reduce the design space, a reasonable amount of inaccuracy is still tolerable to accelerate the design space exploration.

In Chapter 6, we assumed that all of the buses in an architecture candidate are synchronized with a single global clock and its frequency is a reciprocal of memory access time for one word. More sophisticated exploration on these parameters should be addressed as a future work. In addition, even though only the performance metric was investigated in the chapter, the proposed exploration methodology is extensible to consider other metrics such as power consumption, which is currently under development. Another future work is the extension of the proposed methodology to off-chip system, i.e. board level system.

Although only single port shared memory architectures were considered as a design axis, employing other shared architecture, such as multi-port memory or FIFO, might yield better result. This would also require more sophistication on the patterns of accesses to shared memory.
Bibliography


Bibliography


Bibliography


Bibliography


Bibliography


Bibliography


[38] OMI 324 PI Bus, Rev 0.3d. OMI Standards Draft, 1994 Siemens AG.


Bibliography


Bibliography


Bibliography

[57] ARM720T. [Online] Available:
    http://www.arm.com/products/CPUs/ARM720T.html

[58] S. Srinivasan, L. Li, N. Vijaykrishnan, “Simultaneous partitioning and
    frequency assignment for on-chip bus architectures,” in Proceedings of

    design for low-power systems,” in IEEE Transactions on Very Large Scale

    splitting,” in Proceedings of Design Automation and Test in Europe, pp.612-
    616, March 2000.
초 록

시스템-온-칩이 점점 복잡해 지면서 온-칩 통신은 가장 중요한 설계과정중 하나로 인식되고 있다. 특히 시스템 수준의 관점에서 볼 때, 점점 많은 수의 다양하고 복잡한 특성을 갖는 구성 요소들의 집적으로 인해 온-칩 통신량이 점점 증가하고 그 특성 역시 다양화되고 있는 추세이다. 결과적으로 온-칩 통신은 전체 시스템에 대해 성능이나 전력소모, 신뢰성 등의 측면에서 중대한 영향을 가미하게 되었으며, 이로 인해 온-칩 통신에 대해 엄격한 제약을 만족할 수 있는 설계 방법이 요구되고 있다.

본 학위 논문에서는 시스템-온-칩 설계 시 주어진 응용의 온-칩 통신의 특성을 이용하여 통신 구조를 최적화하는 설계 방법론을 제시한다. 다양한 통신 구조 중에서 간단한 구조에 기인한 설계의 편의성으로 인해 현재 가장 널리 사용되는 기반의 통신 구조에 초점을 맞춘다.

그러나 버스 기반의 통신 구조에 국한된 설계 공간만으로도 버스 토폴로지나 버스 개수, 브릿지, 프로세서들의 버스에 대한 할당, 버스 중재 방법, 버스 클럭 속도, 데이터 버스의 폭 등의 다양한 요소들에 의해 그 크기는 매우 커질 수 있다. 따라서 이들의 설계 공간에 대한 효율적인 탐색 방법이 필요하며, 이를 위해서는 빠르고 정확한 성능 예측이 요구
그러나 일반적으로 정확성과 소요 시간은 성능 예측에 있어서 상충하는 조건들이므로, 알려진 성능 예측 방법들은 이 중 하나만을 만족 시킬 수 밖에 없었다. 성능 예측 방법은 크게 정적 예측과 시뮬레이션으로 나누어진다. 정적 성능 예측은 주로 전송되는 데이터의 양이나 전송에 사용되는 채널의 데이터폭 등의 정적인 정보에 의존한다. 이는 빠르게 성능을 예측할 수 있지만 통신 구조상에서의 동적인 경동을 고려하지 못하기 때문에 예측의 정확성이 떨어지는 단점이 있다. 반면, 시뮬레이션은 주어진 입력에 의해 시스템을 직접 실행하여 성능을 예측하므로 결과가 매우 정확한 반면에 이는 엄청난 시간이 걸리는 단점이다. 현재까지 알려진 두 개의 장점을 모두 갖는 성능 예측 방법은 존재하지 않는 것으로 알려져 있다.

제시하는 설계 방법론은 앞에서 설명한 두 가지 성능 예측 방법의 장점을 모두 취하기 위해 전체 탐색 과정을 두 단계로 나눈다. 첫 번째 단계에서는 거대한 설계 공간에 대해 정적 성능 예측 방법을 이용하여 이미 있는 설계 공간을 빠르게 찾아낼 수 있다. 보다 넓은 설계 공간을 다루기 위하여, 버스 토폴로지, 버스 프로토콜, 메모리 할당 등의 다양한 인자를 고려한다. 성능 예측을 위해 큐잉 모델을 이용하는데, 이는 버스 상에서의 프로세서들의 동시 접근에 의한 충돌을 고려하여 정확한 예측을 가능하게 해준다. 또한 다양한 버스 문제 방법과 다중 테스크 환경
에 대한 성능 예측이 고려되었다. 개발된 성능 예측 방법은 다양한 조건의 통신 구조에 대해 시뮬레이션과 비교해 10% 이내의 오차를 갖는 정확성을 가졌고 동시에 수십 배 이상 빠르게 수행된다.

두 번째 단계에서는 트레이스를 이용한 시뮬레이션을 이용하여 첫번째 단계에서 찾아낸 의미 있는 설계 공간들에 대해 정확하게 성능을 검증한다. 이로써 온-칩 버스 구조에 대해 다양한 설계 인자들을 간의 상관 관계를 얻을 수 있다. 제안된 설계 방법론은 4-채널 디지털 비디오 레코더와 OFDM DVT-B 수신기를 위한 동화기에 적용되었다. 제안된 탐색 방법에 의해 얻어진 최종 성능은 초기의 2배에 달하는 것으로 분석되었다. 이는 통신 구조의 설계가 시스템의 전체 성능에 큰 영향을 미칠 수 있음을 보여주는 것이며, 또한 제안하는 설계 방법론이 온-칩 버스 구조의 설계 공간 탐색에 효율적으로 사용될 수 있음을 입증한다.

주어온: 시스템-온-칩, 온-칩 버스 설계, 설계 공간 탐색, 성능 예측, 큐잉 이론, 구조 최적화

학번: 2000-30463