Methodology and A Framework for Efficient Design Space Exploration at A System Level

Seiya Shibata

ABSTRACT

This dissertation presents a system-level design framework for design space exploration of embedded systems.

Design space of embedded systems has been growing. Recent embedded systems have been required to have multiple functionalities in themselves. In order to realize the functionalities under their constraints such as performance, hardware areas and power consumptions, they have been adopting complex architecture which have not only dedicated hardware modules but also multiple processors. Since the performance and costs of embedded systems depend on the mapping of their functionalities onto hardware modules and processors, system designers should find an appropriate mapping efficiently from the large design space.

System-level design is one of key methodologies in order to deal with the large design space. The main purpose of system-level design is to find appropriate architecture of systems including hardware architecture such as the number of processors and dedicated hardware and mapping of functionalities onto them.

In system-level design, system designers generally develop models of systems at high level and evaluate mappings of them. If the evaluation results are not sufficient for their requirements, designers change the mapping or refine the models, and evaluate them again. The designers iterate refinement and evaluation until the models meet their requirements.

Our framework consists of four tools which cover overall system-level design: SystemBuilder-

MP, covalidation environment, system-level profilers and a fast performance estimation tool. The main objective of our framework is to support the design space exploration by iteration of modeling, implementation and evaluation.

Modeling was done in the system description model defined by SystemBuilder-MP. Models are automatically converted into implementations according to a mapping by SystemBuilder-MP. With the SystemBuilder-MP, system designers can obtain implementations of a system without manual works and thus can easily evaluate them. The functionality of the implementations can be validated in short time by the covalidation environment which utilizes a fast Real-Time OS model and an FPGA. Mappings of a system are evaluated by executing the implementations on an FPGA (Field Programmable Gate Array). The system-level profilers are automatically instrumented into the implementations on an FPGA by SystemBuilder-MP and records behavior of them. With visualization of profiles recorded by the profilers, system designers can easily analyze their performance such as parallelism in order to refine them. Moreover, the fast performance estimation tool estimates performance of various mappings fast and accurately from profiles obtained by a few implementations. Using the performance estimation tool, system designers can prune mappings which are obviously unnecessary without implementations of them.

With these tools, designers can explore mappings of a system fast and easily since most of works for implementation and evaluation are done automatically. Therefore efficient design space exploration at a system-level is realized.

This dissertation describes the detail of above tools and evaluates them using some case studies. With the case studies, the smoothness and easiness of design space exploration at a system level are demonstrated.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 BACKGROUNDS

These days, embedded systems are used everywhere in order to make our life convenient. Embedded system means equipment which has computer inside to achieve (or support to achieve) its special purpose. For example, cell phones, which are most important communication tools for us, have computers for handling spoken words and managing phone addresses.

Embedded systems should meet the requirements such as performance, hardware area and power consumption strictly since embedded systems are used at restricted environment. For example, cell phones should be designed to work with low power since batteries drive them and televisions and other consumer electronics are preferred to be small, low power and high performance for convenience.

Moreover, the design term of embedded systems should be shortened in order to meet the time-to-market requirement. Time-to-market means the term from the planning of a product to release of it. In general, the earlier the products are released, the more chance they have in market. Therefore the efficiency of system design is important.

Embedded systems consist of software and hardware. From the view point of embedded system designers, both software and hardware should be optimized to meet their requirements. Therefore, in most case, both software and hardware are newly developed at least partly for each product.

Recent embedded systems have been required to have multiple functionalities in themselves and their design space have been growing. In order to realize their requirements, they have been adopting complex architecture which have not only dedicated hardware modules but also multiple processors. Since the performance and costs of embedded systems depends on the mapping of their functionalities onto dedicated hardware modules and processors, system designers should find an appropriate mapping efficiently from the large design space. In particular, embedded systems generally have strict constraints on execution time (real-time constraints), their hardware area and their power consumption. In order to design embedded systems with required functionalities meeting such constraints, system designers need to evaluate design candidates quantitatively and explore design space (design space exploration).

System-level design is one of key methodologies in order to deal with the large design space. The main purpose of system-level design is to find appropriate architecture of systems including hardware architecture such as the number of processors and dedicated hardware modules and mapping of functionalities onto them.

In system-level design, system designers generally develop models of systems at high level of abstraction and evaluate them. If the evaluation results are not sufficient for their requirements, the designers refine the models and evaluate them again. The designers iterate refinement and evaluation until the models meet their requirements.

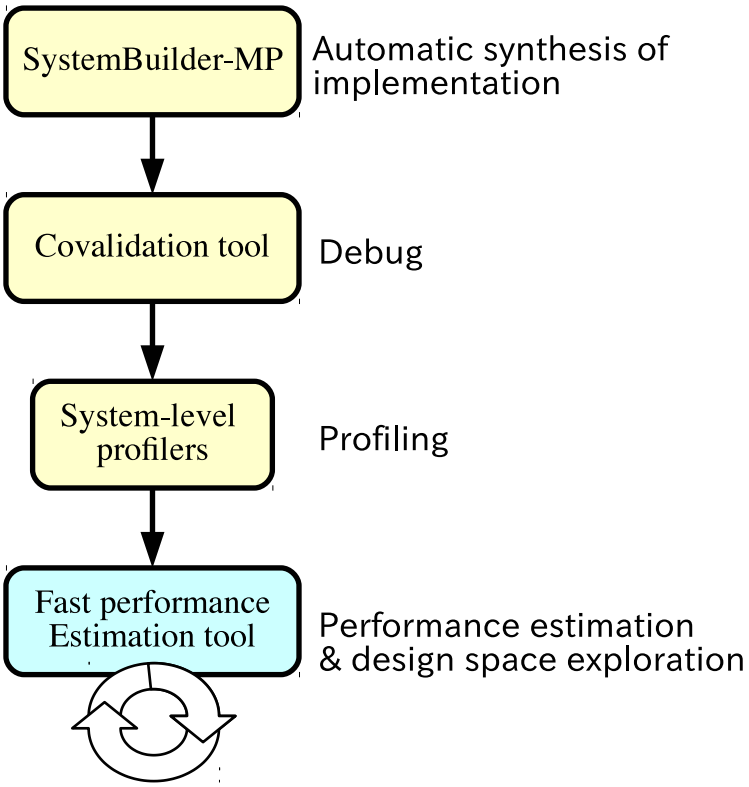


Figure 1.1: Relationships of the four tools in our framework.

1.2 PROPOSED FRAMEWORK

The main objective of our framework is to support the design space exploration by iteration of modeling and evaluation. Our framework consists of four tools which cover overall system-level design: SystemBuilder-MP, covalidation environment, system-level profilers and a fast performance estimation tool. Fig. 1.1 shows the relationships of the four tools.

SystemBuilder-MP is an automatic synthesis tool. It defines a system description model and provides automatic synthesis capability of them. In the system description model, system designers describe system functionalities and communications among them as processes and channels. As a result of automatic synthesis, system designers can obtain im-

plementation descriptions of systems for various mappings easily. In particular, the system description model is defined for expressing parallel behaviors. SystemBuilder-MP can convert the model into implementations with multiple processors and dedicated hardware modules. In order to utilize the parallelism of processors and hardware, SystemBuilder-MP can explore not only mapping of functionalities but also bus interface architecture. With SystemBuilder-MP, system designers can find architecture where shared resources such as buses and memories are efficiently used. The implementation descriptions can be executed on both simulation tools and FPGA (Field Programmable Gate Array).

Covalidation is a validation of software and hardware with their cooperative execution. Our covalidation environment provides fast validation and debug capabilities of system functionalities by executing both of them in high speed. In order to execute software and hardware fast, our covalidation environment uses a Real-Time OS (RTOS) model and an FPGA, respectively. The RTOS model can be executed on a host PC natively. The FPGA is connected to the host PC and communicates with software on the RTOS model.

Implementations of systems executed on an FPGA are called FPGA-based prototypes. FPGA-based prototypes are appropriate for evaluation of systems since they can be executed fast and accurately. In addition to the fast evaluation, our framework provides system-level profilers for FPGA-based prototypes which record behavior of systems. With the profilers, system designers can observe parallelism of system components using an FPGA and analyze performance of them. Since the profilers are automatically instrumented into FPGA-based prototypes, there is no need for manual works to use them.

In order to accelerate design space exploration, our framework also provides a fast performance estimation tool. With the fast performance estimation tool, system designers can evaluate a number of mappings on a host PC without synthesizing FPGA-based prototypes. In particular, our performance estimation tool has a stochastic model of bus arbitration delay. With the fast performance estimation tool, system designers can explore mappings

considering the effect of bus arbitration delay which have large influence on performance of recent multiprocessor systems.

By using our framework with the four tools, system designers can design embedded systems with multiprocessor and dedicated hardware modules efficiently.

This dissertation describes the detail of above tools and evaluates them using case studies. In the case studies, the smoothness of design space exploration and easiness of design analysis using the framework are demonstrated.

1.3 OUTLINES OF THE DISSERTATION

The organization of this dissertation is as follows.

First, Chapter 2 summarizes the system design methodologies including past ones and current system-level design one, and shows stand point of our framework. Chapter 3 describes related works of this dissertation. Next, Chapter 4 describes base tools of tools proposed in this dissertation and a case study on MPEG-4 decoder design as a preliminary. Then, Chapter 5 describes the detail of SystemBuilder-MP and Chapter 6 proposes covalidation environment accelerated by an RTOS model and an FPGA. Chapter 7 presents system-level profilers. Chapter 8 proposes a fast performance estimation tool which accelerates design space exploration effectively. Finally, Chapter 9 concludes this dissertation with summary.

CHAPTER 2

SYSTEM-LEVEL DESIGN

This chapter explains the backgrounds and basics of system-level design methodology, and an overview of our framework.

Since the construction of embedded systems has been the combination of software and hardware throughout several decades, their general design flow has also remained. However, as the complexity of embedded systems have grown, system design methodologies have been developed to keep efficiency of the general design flow. And design space exploration at a system level is currently receiving attentions as a solution for recent complex system design.

In this chapter, first, 2.1 briefly shows the general design flow of embedded systems. Then 2.2 describes past design methodologies developed to make the general design flow efficient and shows the problems of them on current embedded system design. In 2.3, an overview of design space exploration at a system level and approaches to it are shown. Finally 2.4 illustrates the overview of the proposed framework with the relationships with the system-level design.

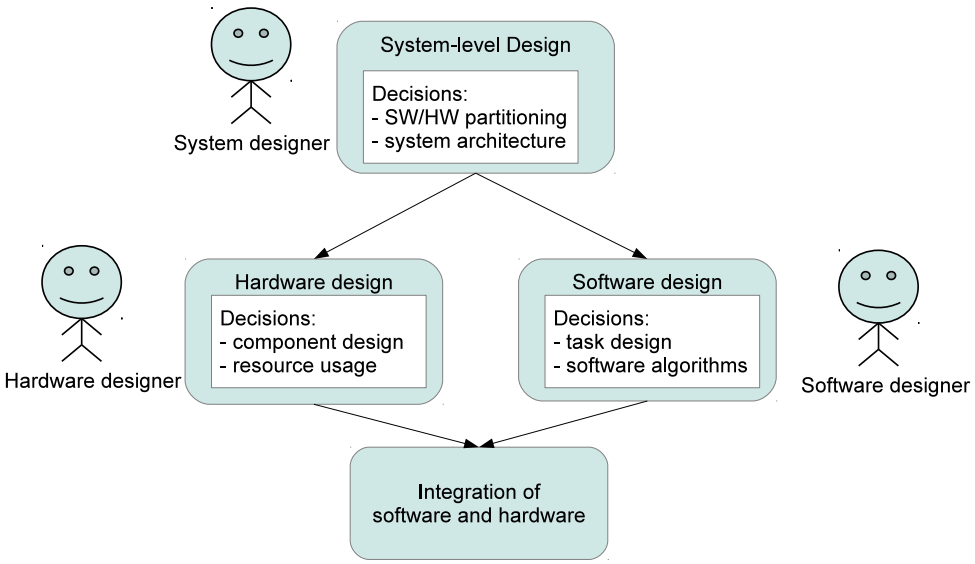


Figure 2.1: General design flow of embedded systems.

2.1 GENERAL DESIGN FLOW OF EMBEDDED SYSTEMS

In general, embedded system design after requirement decisions consists of four phases: system design, hardware design, software design and integration (Fig. 2.1)

First, system designers decide overall system architecture and decide software/hardware partitioning. After the partitioning, hardware designer and software designer decide detail of hardware and software, respectively. Finally manufactured hardware and developed software are integrated. In order to design systems which meet their requirements in a short time, each phase of design should be efficient.

In all phases, decisions are made by designers. System designers decide software/hardware partitioning and construction of hardware such as the kinds and numbers of processors and dedicated hardware. Hardware designers decide hardware resources and usage of them. Software designers decide structure of software such as tasks.

In this design flow, the earlier the phase is, the more influence the decisions have on performance/cost of the design. Therefore decisions of system designers have the largest

effects on final products and are the most important in the flow.

2.2 DESIGN METHODOLOGIES

So far, methodologies of embedded system design have been proposed in order to make the general design flow efficient. Hereafter, we briefly summarize the history of the design methodologies.

2.2.1 PAST DESIGN METHODOLOGIES

In traditional system design (illustrated in Fig. 2.2), the most important decision on system-level design was software/hardware partitioning since traditional embedded systems used to be relatively simple ones which have only a single processor and dedicated hardware modules. Since the scale of systems were small enough, software/hardware partitioning was generally done by skilled system designers with their experience and intuitions. After the software/hardware partitioning, first hardware designers developed the hardware ("HW design/development" in the figure) and then software designers started their development on the manufactured hardware considering integration with the hardware ("SW design/development (including integration)" in the figure) as described in the literature [1].

As the scale of systems had grown gradually, software and hardware for such systems had become complex. Thus the efforts to make the system design flow efficient were first conducted on techniques on software and hardware design (shown in Fig. 2.3). In order to develop large and complex hardware in short time, abstraction level of hardware design had been raised from the transistor level to the register transfer level (RTL). For the hardware design at RTL, RTL languages such as VHDL and Verilog-HDLs, their synthesis tools and simulation tools were proposed. In the RTL languages, hardware designers can describe the functionality and structure of hardware as construction of registers and the

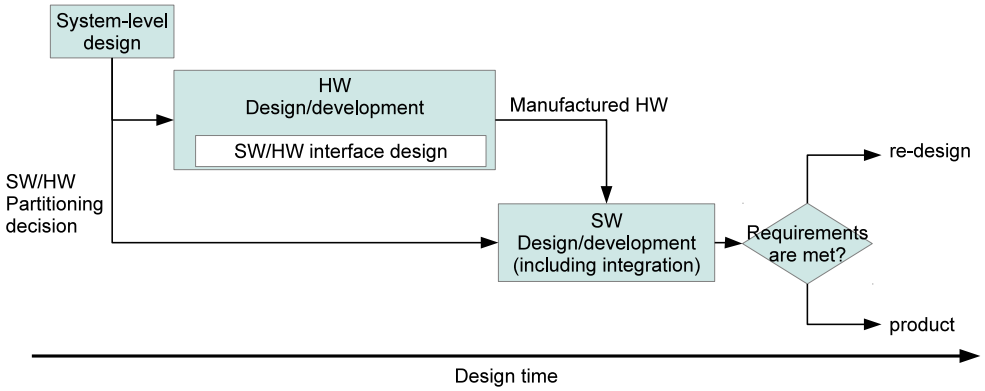


Figure 2.2: Traditional system design flow.

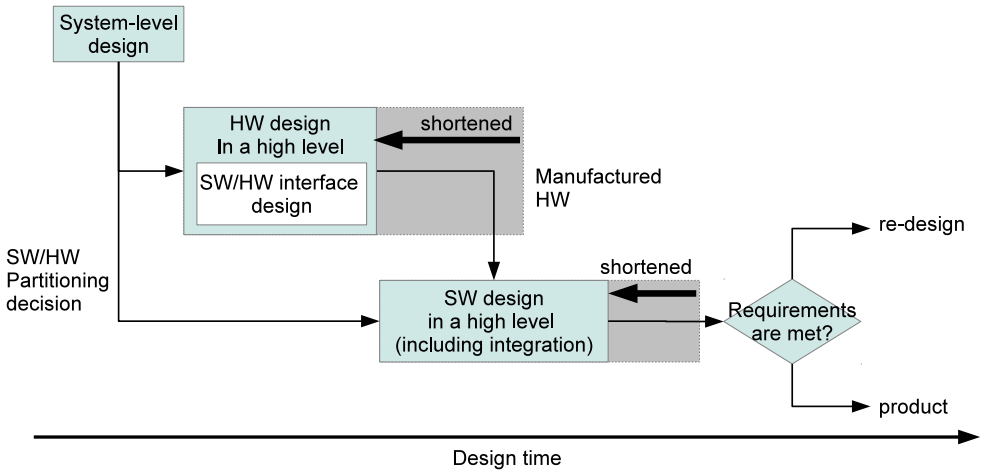


Figure 2.3: Raises of abstraction level in software and hardware design.

transitions of signals among registers. Moreover, in last decades, behavioral synthesis tools had been proposed to raise the abstraction level of hardware from RTL to behavioral level described by behavioral languages such as the C, SystemC and SpecC languages. At the behavioral-level hardware design, hardware designers describe only the functionalities of the hardware. Behavioral synthesis tools automatically generate the structure of the hardware for the functionalities. Same as hardware design, abstraction level of software was also raised from a binary level to a mnemonic level and a structured level. The mnemonic level was achieved by assembly languages and the structured level was achieved by the

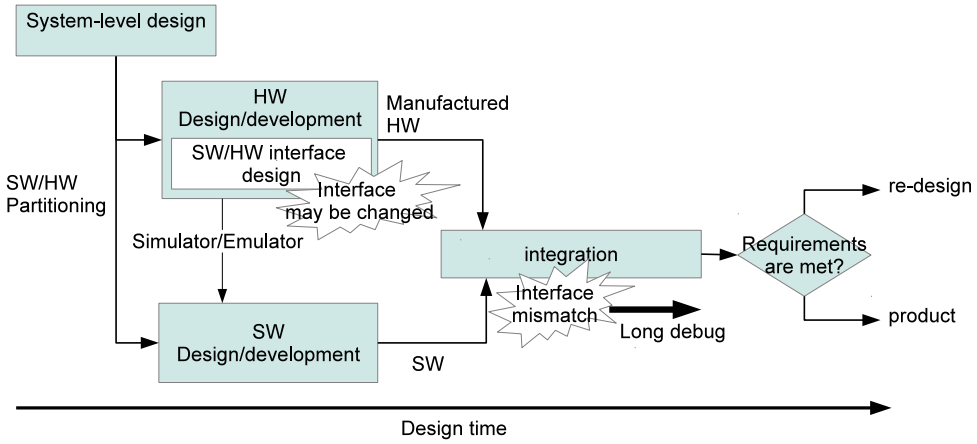


Figure 2.4: Concurrent design methodology

C language. By the promotion of abstraction levels of software and hardware design, design term of software and hardware were shortened dramatically. However, the order of development of software and hardware remained untouched since the languages were not developed considering each other.

In 1990's, concurrent design of software and hardware were proposed as a promising technique for achieving short design term. For the concurrent design, RTL simulation tools are used to execute software before manufacturing the hardware (illustrated in Fig. 2.4). RTL simulation tools can simulate cycle-level behavior of hardware on a host PC and thus they can execute software on them. The appearance of FPGAs (Field Programmable Gage Arrays) also helped the execution of software on hardware before manufacturing. FPGA is a configurable hardware which can emulate cycle-level behavior of hardware. With FPGAs, hardware can be executed accurately at faster speed than RTL simulators.

The concurrent design methodology using simulators/emulators of hardware, however, involved a problem which was due to the mismatch of interface between software and hardware. This is caused by miscommunication among software designers and hardware designers. Since the interface between software and hardware were mainly defined by hardware designers and the definition was often changed for performance and cost issues, the

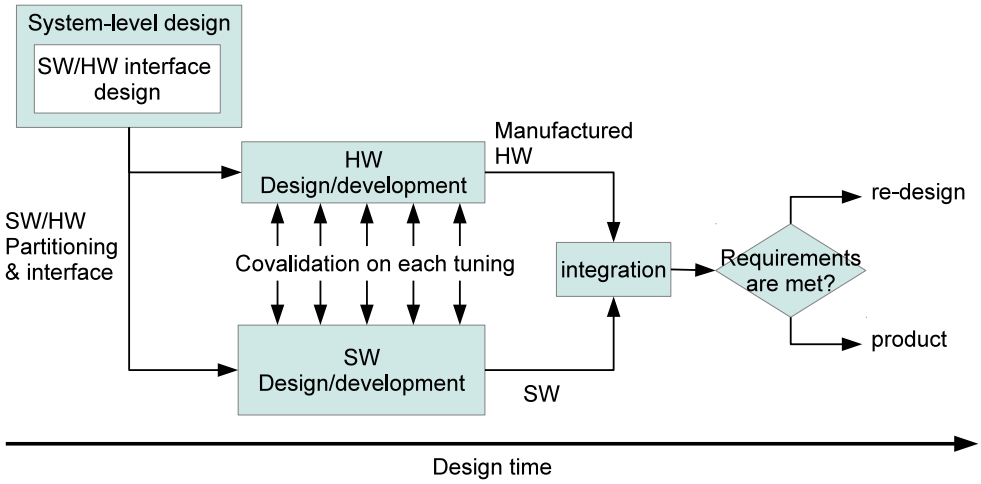


Figure 2.5: Codesign flow.

miscommunication of the changes caused mismatch of the interface on integration phase and resulted in bugs. Long time might be spent for the debug of them.

In order to solve such miscommunication among software and hardware designers, codesign methodology was proposed (illustrated in Fig. 2.5) [2][3][4]. In the codesign methodology, software and hardware are designed and developed concurrently. The interfaces among software and hardware are strictly defined at system-level design phase and the interfaces are kept during their development. Covalidation is a technique which confirms the behavior of software and hardware including their interfaces [5]. Cosimulation tools are a kind of realization of covalidation, which consist of a set of simulation tools which can execute both software and hardware on a host PC. By confirming that the interfaces are kept on each tuning of them using covalidation, bugs on the integration phase can be decreased. With codesign methodology with covalidation, designers can shorten design term avoiding the drawbacks caused by miscommunication.

2.2.2 CURRENT PROBLEMS

Recently, embedded systems have been more and more growing their complexities with the growth of their functionalities and architecture to the extent of codesign only methodology cannot deal with. The problems have been rising on the system-level design phase.

Moreover, time-to-market pressure has been becoming hard in order to win the growing market. Therefore the past design methodologies shown in 2.2, which depend on skills and intuitions of system designer, have become hard and impossible to handle such complexity of recent systems.

One of the factors which make the complexity of system design high is the usage of multiprocessor architecture. A System-on-a-Chip with multiprocessor is called an MPSoC [6]. MPSoCs are promising solution for improving processor performance while keeping the power consumption of them low. However, the design of MPSoCs caused another problem: exploration of an appropriate mapping of the functionalities onto processors. In the exploration, system designers should consider not only software/hardware partitioning but also mapping of functionalities onto the various kind and number of processors. In particular, the mapping is important in the embedded system where functionalities of them are mapped onto processors statically in order to meet real-time requirements such as predictability and verifiability.

Memory architecture is another problem on system design. There are several kinds of memory modules which differ their read/write latencies and their cost. In general, memory modules with small latencies (hence they are fast) are expensive, and in contrast those with large latencies (hence they are slow) are cheap. Therefore system designers should consider the trade-offs between performance and cost on memory architecture by exploring the mapping of data buffers in the functional models onto memory modules.

With the complex problems of mappings of functionalities onto MPSoCs and data

buffers onto memory modules (hereafter, mappings), it is possible to fail to meet the requirements on integration phase with any of the past methodologies such as traditional and codesign methodologies since mapping decisions are not evaluated on system-level design phase. Failure of meeting the requirements may cause re-design and re-development of the system. Moreover, the manufacturing cost of the system must also be much larger.

In order to avoid failure found on integration phase, the quantitative evaluation of mappings at system level has become important to decide appropriate mappings.

2.3 DESIGN SPACE EXPLORATION AT A SYSTEM LEVEL

Design space exploration (DSE) at a system level is one of key methodologies in order to deal with the recent large design space. The main purpose of DSE at a system level is to find appropriate architecture of systems including hardware architecture and mapping using a quantitative evaluation.

In DSE at a system level, system designers generally develop models of a system at a high level and evaluate them (illustrated in Fig. 2.6). If the evaluation results are not sufficient for their requirements, designers refine the models and evaluate them again. The designers iterate refinement and evaluation until the models meet their requirements. Since the appropriateness of mapping is confirmed at a system level, the possibility to find failure at the integration phase should be much lower.

Models of systems are generally defined as processes and channels which represent functionalities and communications among them. Processes are units of mapping exploration onto hardware architecture.

Hereafter, we summarize past approaches to system-level design.

In early 2000s, mapping exploration algorithms were proposed which schedules execution timing of processes and allocation of them as an extension of behavioral synthesis

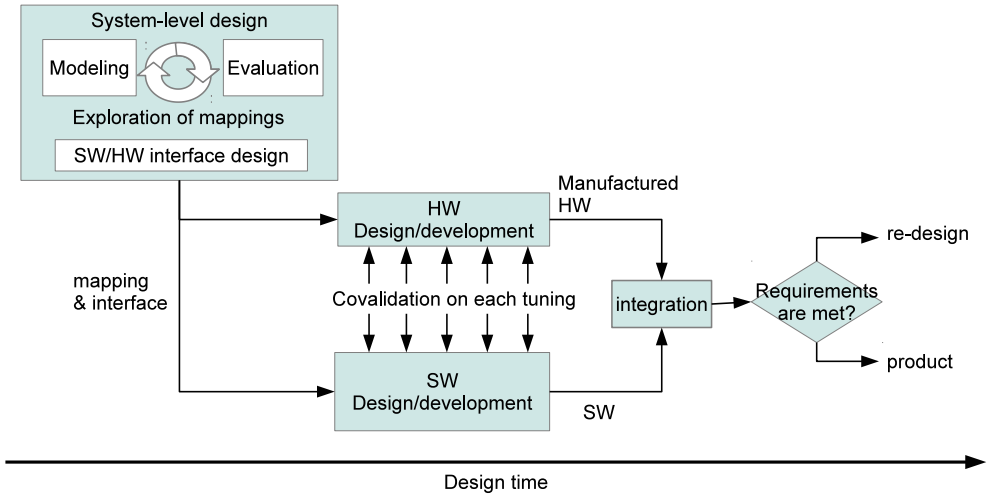


Figure 2.6: Design space exploration at a system-level design.

algorithms [7][8]. In these algorithms, processes were only abstract models which have parameters such as execution time and costs. Although such algorithms could produce a solution which may meet its requirements, the solution often resulted in no appropriate solution for the actual implementation of them. The reasons of it were lacks of consideration of variation on execution time of processes and overheads on communication among processes, that is, channels. Moreover, the consideration of both processes and channels in such algorithms resulted in complex formulation which is hard to solve.

After that, estimation-based approaches and implementation-based approaches were proposed in order to evaluate and explore mapping considering effects of both processes and channels on performance and costs.

Fig. 2.7 shows a typical flow of estimation-based approaches. In estimation-based approaches, models which represent performance and/or costs are developed from abstract models of a system [9][10]. Similar to the exploration algorithms, such performance/cost models often have only parameters such as execution time. Parameter setting of the performance/cost models is important since the accuracy of estimation depends on it. In addition to parameters of processes, estimation methods should consider effects of channels. Some

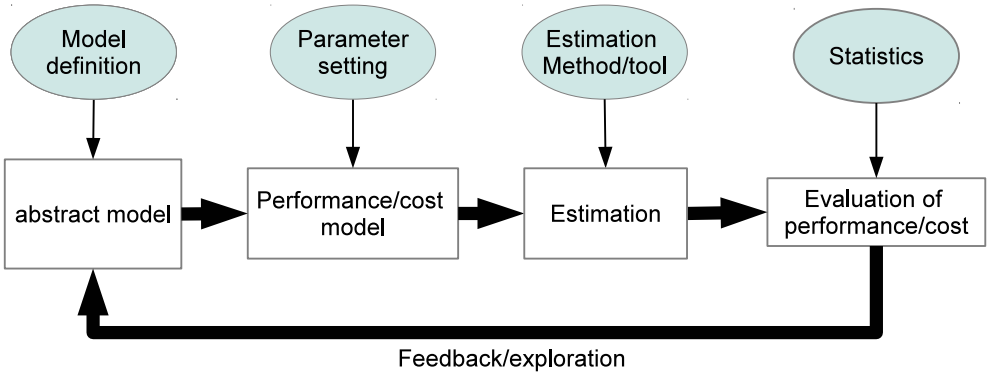


Figure 2.7: A typical flow of estimation-based approaches.

estimation methods use simple calculation and others use abstract simulation method to estimate performance and costs. With estimation results, system designers evaluate mappings and explore the appropriate mapping. For the purpose of mapping exploration, estimation methods should be fast and should keep fidelity, that is, the validity of relative comparison of mappings on their results. Instead, they can involve some amount of error as long as the fidelity is kept.

The estimation-based approach requires some tools to obtain parameters such as execution time. Since the accuracy of estimation depends on the parameters, most estimation methods use precise simulation/emulation methods to obtain such information. Precise simulation/emulation methods, however, spend long time to be prepared and executed. Therefore the estimation-based approach should be developed considering the trade-off between accuracy and estimation time.

One of problems on application of estimation-based approach is the method to obtain the information for setting parameters. Simple solution of this problem is to set parameter according to past experiences. However, such simple solution cannot be applied for newly developed functionalities. Also, it may lack consideration of variation on execution time similar to the mapping exploration algorithms.

Fig. 2.8 shows a typical flow of implementation-based approaches. In implementation-

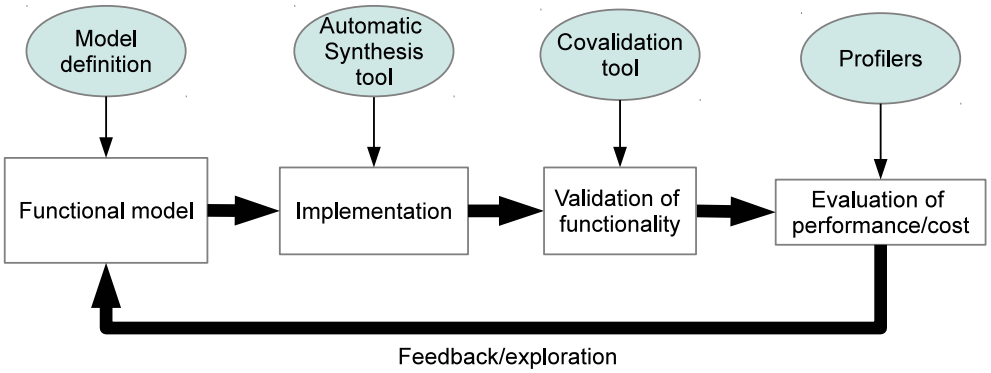


Figure 2.8: A typical flow of implementation-based approaches.

based approaches, executables were developed based on abstract models. Typically, executables means implementation of systems for cycle-level simulators and/or FPGAs. Performance and/or costs are evaluated by execution of them. Unlike estimation-based approaches, the evaluation results are obtained from the execution without any lack of consideration. Instead, the implementation-based approaches generally require automatic synthesis of implementation from abstract models since manual development of implementations for various mappings may result in long design time and be error-prone. If the processes are described in behavioral languages such as the C language, they can be converted into software binaries and hardware descriptions using compilers and behavioral synthesis tools, respectively. Moreover, automatic synthesis for system-level design is required to generate implementation of channels. Channels are generally converted into glue descriptions/logics for processes. In this approach, though system designers need to describe processes in behavioral language, they can evaluate the performance and costs of a mapping accurately by simulation or execution of the FPGA-based prototypes.

In order to support the implementation-based approach, some tools are required. One is an automatic synthesis tool which generates implementations as described above. System modeling method as input for the automatic synthesis is also important for efficiency of system design. Moreover, a covalidation tool which support to verify the functionality of

generated implementation of a system is required for smooth development of a system. The covalidation tool should be fast and easy to use with integration of an automatic synthesis tool. Performance analysis tools are also useful for exploration of mappings. Since the number of mappings reaches millions or more, performance analysis of an FPGA-based prototype may help decide next mapping to be explored.

One of problems on application of implementation-based approach is long synthesis time of FPGA-based prototypes. Since behavioral/logic synthesis of hardware is computation intensive, such synthesis may take minutes or hours. If we assume that there are a thousand of mappings to be explored, implementation-based approach may take more than one month. In the design of MPSoCs, the number of mappings actually reaches millions or more. Therefore some sort of technique is required to prune mappings which are implemented and evaluated on FPGA.

2.4 PROPOSED FRAMEWORK OVERVIEW

In order to develop an efficient system-level design methodology which do not depend on the skills and intuitions of system designers, this dissertation presents a system-level design framework which consists of four tools: SystemBuilder-MP, covalidation environment, system-level profilers and a fast performance estimation tool. With this framework, system designers can explore large design space in short time with quantitative evaluation and obtain FPGA-based prototypes.

Our framework is constructed by a combination of implementation-based approach and estimation-based approach. By the combination of two approaches, our framework solves the disadvantage of the two approaches and builds the smooth design space exploration methodology. In concrete, SystemBuilder-MP, the covalidation environment and the system-level profilers are tools for implementation-based approach. They help sys-

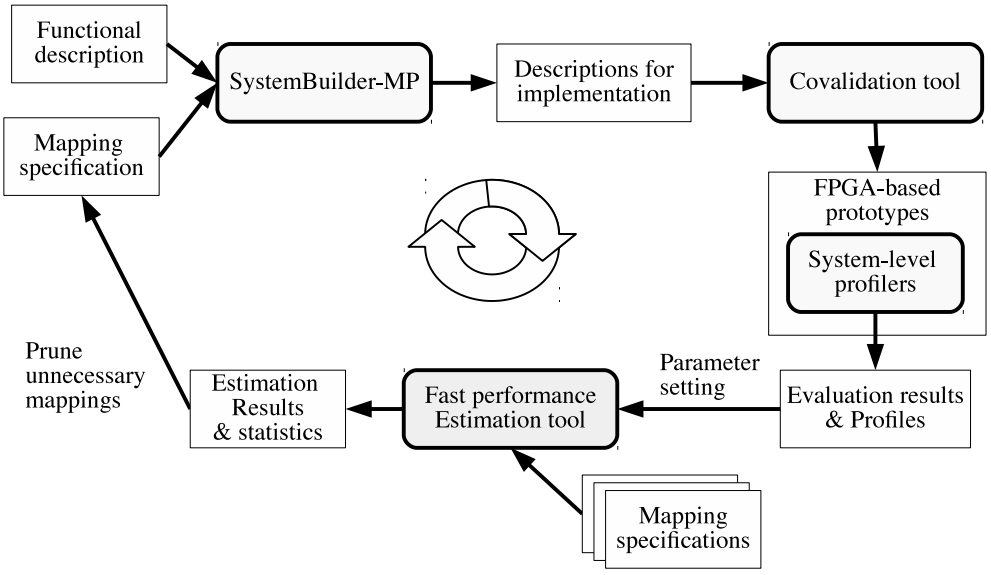


Figure 2.9: System-level design flow of our framework.

tem designers obtain FPGA-based prototypes and their profiles. On the other hand, the fast performance estimation tool is a tool for estimation-based approach. With the three tools for implementation-based approach, system designers can obtain information for setting model parameters for estimation-based approach easily. And with the tool for the estimation-based approach, system designers can evaluate most mappings without FPGA-based prototypes. Thus system designers can reduce the number of mappings which are evaluated by synthesizing the FPGA-based prototypes. The reduction of the number of mappings contributes to shortening the design term.

This section shows overview of the framework.

Fig. 2.9 shows the system-level design flow supported by the four tools.

First, system designers describe the functionalities of systems (“Functional description” in Fig. 2.9), and synthesize FPGA-based prototypes using our system-level design tool, named “SystemBuilder-MP”, according to “Mapping Specification”. SystemBuilder-MP was developed on the basis of our past work “SystemBuilder”.

Covalidation environment is used to debug in the development of system description. Unlike other cosimulation tools, our covalidation environment uses not only simulators but also an FPGA. FPGA is used to accelerate execution of dedicated hardware modules. Moreover, an RTOS model is used to execute software faster than ISSs (instruction set simulators). With both the RTOS model and the FPGA, the covalidation can be done in short time.

Next, designers obtain a few "Profiles" from FPGA-based prototypes. In order to obtain the profiles, SystemBuilder-MP automatically instruments "System-level profilers" in the FPGA-based prototypes. With the system-level profilers, designers can obtain the profiles of parallel behavior of processes and memory accesses.

After profiling of a few mappings, designers can explore large number of mappings (specified in "Mapping specifications (for estimation)") with our fast performance estimation tool. In the estimation method, parameter settings were done automatically from inputs for SystemBuilder-MP and profiles obtained by a few executions of FPGA-based prototypes. Since estimation of a mapping finishes in seconds, designers can explore large number of mappings in short time. Even though the performance estimation results involve some amount of errors, system designers can prune mappings which are obviously unnecessary before generation of implementations of them.

By the seamless cooperation of four tools in our framework, implementation of FPGA-based prototype, debug, instrumentation of system-level profilers and performance estimation can be done with few manual efforts. Therefore our tool framework realizes the efficient flow of design space exploration of design space exploration at a system level.

CHAPTER 3

RELATED WORKS

Several tools have been researched and proposed for implementation-based evaluation approach at system-level design.

3.1 AUTOMATIC SYNTHESIS TOOLS

Automatic synthesis tools approach the mapping exploration problem by automatically generating implementation of system from abstract definition of systems.

One of problems on design space exploration is time to spend to develop implementations. Design space exploration at high level is the first phase of system development and it should be finished as soon as possible to meet short time-to-market requirements. However, manual development of all implementations may be error-prone and time-consuming. In particular, difference of mapping involves difference of communication among functionalities. Thus the number of implementations which should be developed increases exponentially depending on the number of functionalities and communication among them.

By automatic synthesis, designers become not to suffer from change of implementation according to change of mapping. Automatic synthesis tools generate implementation of

systems from abstract representation of them.

For example, several system-level design tools have been developed and proposed for automatic synthesis of systems from abstract models [11][12][13][14]. Literature [15] surveyed such tools and compared them. SystemBuilder-MP, which is proposed in 5, is placed in this category.

These tools supports design methodologies where designers starts design of a system from describing it at a high-level of abstraction and gradually refine it into lower-levels of abstraction and obtain final products. SCE proposed uses SpecC [16] as an abstract model of systems [11]. KPNframework uses "Kahn Process Network" [12] and PeaCE uses an original model called SPDF [13]. SystemC [17] is a major model for system-level design [18] and is adopted by SystemCoDesigner [14].

Tools proposed in [12] and [13] automatically synthesize not only processes but also original task scheduler instead of RTOSs in order to handle multiple processes on processors. Although the use of the original task scheduler enables the system to shorten overhead of schedule, systems without RTOSs obviously lack other capabilities such as hardware abstraction and communication among processors. Since SystemBuilder-MP uses RTOSs based on ITRON specification which is popular specification of RTOS in Japan, systems synthesized by SystemBuilder-MP can use various hardware modules including various kinds of processors. Moreover, the use of ITRON-based RTOSs enables designers to integrate synthesized systems with other ITRON-based systems.

ELEGANT is a system-level design framework which is similar to SysmtemBuilder-MP [19]. It uses SpecC as a language for describing system functionalities. Automatic synthesis of communication is done by SER [20] and that of processes is done by behavioral synthesis tool CyberWorkBench[21]. It uses combination of commercial simulation tools for verification and evaluation of systems, and it does not use FPGA-based prototypes. Since ELEGANT does not use FPGA-based prototypes and use slow (but accurate)

simulation tools, it may take long time to verify and evaluate multimedia systems which needs vast iteration of execution with various kinds of inputs.

ImpulseC/CoDeveloper[22] is an automatic synthesis tool for FPGA. It supports software/hardware partitioning and synthesis of implementation onto FPGA. However, the main objective of ImpulseC/CoDeveloper is design of dedicated hardware intensive systems. It does not support synthesis of multiprocessor architecture.

Some of system-level design tools also include system evaluation method such as simulators and estimators described below and form system-level design frameworks.

3.2 COVALIDATION TECHNIQUES

Validation is to check whether a system has bugs or not. Validation methods are divided into two kinds: test methods and formal methods.

In the test methods, designers feed various inputs to their design and check that the outputs are expected values. Although test methods can check most case of behavior of the design easily with a number of inputs, they may miss rare cases and therefore they cannot completely prove that the design have no bug.

In the formal methods, designers can obtain theoretical proofs that the design does not have a certain bug using mathematical models of their design. In contrast to test methods, they do not miss rare cases. However, the complexity of mathematical models of the design increases exponentially according to the complexity of the design. Thus application of formal methods is currently limited to small designs.

In this dissertation, we focus on validation tool for the test methods since the purpose of our validation is to check that the FPGA-based prototype correctly works with some inputs for performance evaluation. The mathematical proofs that a system has no bug are not required.

As a matter of course, the formal methods are important in embedded system design. Ideally, formal validation of functional descriptions of a system should be applied before the use of our framework. However, it is not a methodology for design space exploration but for correct development. Therefore we left the formal methods out of scope of this dissertation.

Covalidation is a validation method of a system which has software, hardware and communications among them. Hereafter, we summarize covalidation techniques proposed in past.

Gerstlauer et al. proposed generic RTOS models in system-level description languages (SLDL) for cosimulation of hardware and software including RTOS [23]. After cosimulation with such models represented in SLDL, system designers select a real RTOS and prepare final implementations of hardware. Since most of such generic RTOS models support a minimal set of the service calls, the designers need to replace the service calls of the generic RTOS in application software with those of the real RTOS. However, replacement of the service calls is time-consuming and may embed errors into application software. Those errors can hardly be found until cosimulation using an ISS (instruction set simulator). Note that cosimulation with the ISS is very slow, though some advanced techniques (e.g., virtual synchronization [24], the use of SystemC [25] and an RTOS model [26]) are studied for acceleration. Moreover, since final implementations of hardware may behave different from the models, software designers must perform covalidation again with other covalidation environment.

Our covalidation environment can reduce such drawbacks because it can handle not only models but also final implementations and supports smooth transition among them.

3.3 PERFORMANCE EVALUATION METHODS AND TOOLS

Major promising performance evaluation methods are cycle-level simulations and FPGA-based prototypes. Using traditional simulation method for hardware, cycle-level simulation of an overall system can be done in a host PC. However, cycle-level simulation of a system is very slow since they accurately calculate behavior of all of registers and wires in the system on each cycle. Such simulation is said to take $1000\times$ longer time than actual systems in the literature [27].

One of solution to speedup the evaluation is use of FPGAs. By implementing all system components such as processors, dedicated hardware modules, memory modules and interconnections among them in an FPGA (or a set of FPGAs), system designers evaluate overall systems.

Although evaluation methods using FPGAs are much faster than cycle-level simulations, it has some problems. One is limitation on observability of them. Lack of observability means that designers cannot analyze the reason of the evaluation results. Without analysis, it is difficult to improve systems. In order to observe internal behavior of them such as value transitions on registers, they need extra mechanisms for recording such internal behaviors.

By using IPs for debugging hardware provided by FPGA vendors (ChipScope[28], SignalTap[29]), designers can observe internal signals of an FPGA and analyze behavior of the system in detail. These approaches, however, need expertise of hardware and manual modification of the system (hence error-prone) for profiling.

Valle et al. proposed an environment on an FPGA for profiling software which is executed on multi-processor systems [30]. In their environment, clock inputs for the system under profiling can be controlled and the accuracy is guaranteed. Their environment, however, cannot handle dedicated hardware modules. Nunes et al. proposed a profiler construc-

tion for multi-FPGA systems with high-level descriptions of systems[31]. Their approach has a similar concept with ours in assuming that systems are developed using specific communication channels between functionalities and the profilers are developed to cooperate with the channels. However, their profiler needs manual instrumentation.

As another evaluation solution, Nanjundappa et al. proposed fast SystemC simulation on GPUs[32]. The use of GPUs also involves the same problem of FPGAs on observability of them.

In contrast with above tools, our profilers record traces of concurrent MPSoC systems on an FPGA, achieving both high accuracy and short execution time. System designers can profile the large number of design alternatives easily with automatic instrumentation of the profilers.

3.4 FAST/ABSTRACT PERFORMANCE ESTIMATION TOOLS

In order to realize efficient design space exploration, fast and abstract evaluation methods have been proposed recently.

One of them is annotation-based method. In annotation-based method, simulator executes behavioral descriptions such as C programs which have annotations of cycles needed to execute them. In most case, such methods annotate execution cycles to each basic blocks of C programs[33],[34]. Simulators for annotation-based method manages system time (cycles) and advance the time for each time the simulator execute the basic block with annotated value. For annotation-based method, several techniques have been proposed to calculate cycles to be annotated.

More abstract method is trace-based simulation method. Trace-based simulation method uses traces recorded on other simulation method such as cycle-level simulation or FPGA-based prototype execution. Using traces, trace-based simulation method construct simula-

tion model for other situations of the same system and evaluate performance of them.

There are many approaches on trace-based simulation method.

ARTS [35] and TAPES [36] are system-level performance estimation frameworks. ARTS is a framework for modeling and simulating MPSoCs. Given profiles of tasks to be executed on processing elements, ARTS simulates communications among tasks and calculates performance numbers. TAPES provides a retargetable simulation framework with a given profile of the system functionality. These frameworks assume that profiles of the system at a system level are given prior to their simulation, therefore the accuracy of their simulation depends on the accuracy of profiles. Moreover, the focuses of these works are on qualitative analysis such as scalability for multiprocessor systems, and accuracy of them were not mentioned.

The approach proposed in [37] extracts functional characteristics from high-level models of systems, and explores a large number of design candidates at an implementation level considering the functional characteristics and hardware characteristics. Although their approach achieved design space exploration with high accuracy in a short time, it requires a database containing detailed hardware characteristics of target architecture to be developed. Moreover, their estimation method only considers logic level architecture inside a processor and a hardware module such as adders, and do not consider multiprocessor architecture and their parallelism.

The approach proposed in [38] provides performance estimation method where performance parameters are obtained from execution results of instruction set simulators (ISSs). Their approach considers multiprocessor systems and calculates accurate execution time of systems. However, the use of ISSs causes relatively slow estimation speed, and they do not consider communication time among processors.

The basic concept of our performance estimation method is similar to the work by Ueda et al. [39]. However, the focus of their approach is comparison of performance on different

bus topologies with given IP components. Execution time of IP components is assumed to be given by IP database and to be constant. Therefore, their approach cannot accurately estimate performance of systems where some of IPs are activated several times and their execution time vary on each of their activation. Also, they do not mention about how to develop the IP database.

Moreover, most of these performance estimation methods do not consider RTOSs (scheduling times) and interrupts (interruption handling times) which are often used for developing embedded software.

Most of performance estimation methods at a system level do not consider the bus arbitration delay on simultaneous accesses for shared resources such as memory modules since the strict consideration of arbitration delay requires a cycle-level simulation of a whole system using tools such as SoCDesigner[40] and Platform Architect [41]. Some researches were conducted on stochastic modeling of bus arbitration for fast and approximately accurate performance estimation. Bobrek et al. proposed a statistical regression model for estimation of bus arbitration delay [42]. The regression model consists of three parameters which represent the characteristics of memory accesses by a system. The construction of the regression model requires a certain amount of executions of the system for training the model.

Our performance estimation tool has stochastic models of bus arbitration delay which do not need any training. With the models, the effect of conflicts on buses and memory modules can be efficiently considered in the performance estimation.

CHAPTER 4

BASE TOOLS AND MPEG-4 DECODER DESIGN

In this chapter, we show an brief overview of an automatic synthesis tool, cosimulation environment and a case study of MPEG-4 decoder system design. The automatic synthesis tool, named SystemBuilder[43], and cosimulation environment[44] were developped as prior works of SystemBuilder-MP and covalidation environment, respectively . In order to make this dissertation self-contained, we describe about SystemBuilder and the cosimulation environment here.

The case study on MPEG-4 decoder system design was conducted using SystemBuilder and the cosimulation environment. The problems found in this case study was solved by tools proposed in this dissertation.

This chapter is organized as follows. First, Section 4.1 explains a brief overview of SystemBuilder and Section 4.2 shows an overview of the cosimulation environment. Then Section 4.3 presents a case study on MPEG-4 decoder system design. Section 4.4 evaluates effectiveness and problems of SystemBuilder clarified through our case study, and Section 4.5 concludes this chapter.

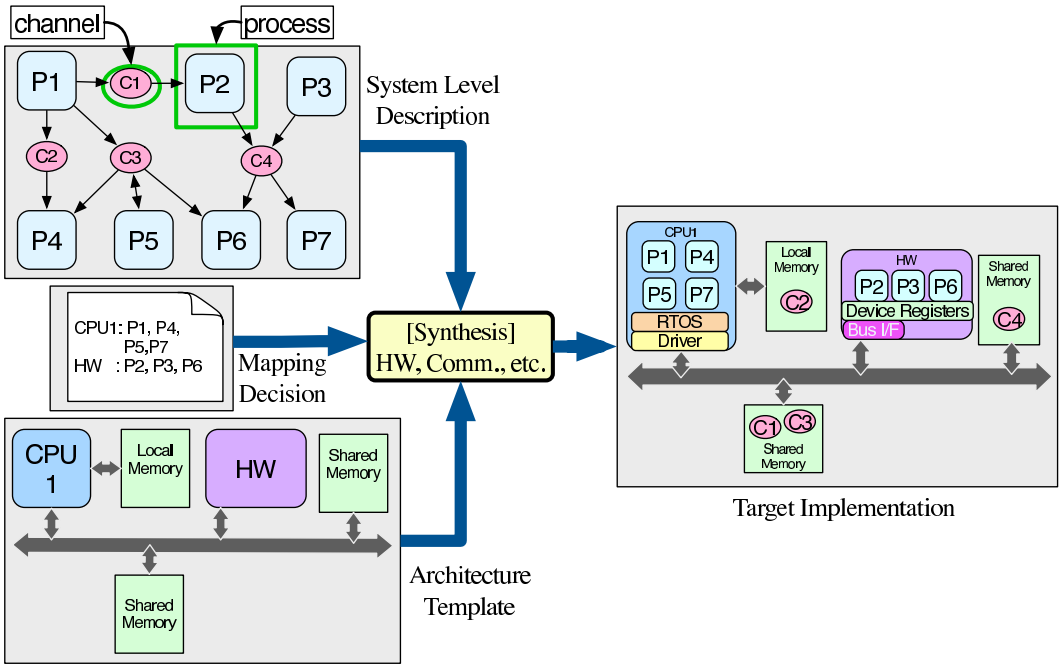


Figure 4.1: design flow with SystemBuilder.

4.1 SystemBuilder

In this section, we show a brief overview of SystemBuilder[43].

4.1.1 Input description

Figure 4.1 shows the design flow of SystemBuilder. SystemBuilder takes *System-Level Description* (*SLD*, hereafter) and an architecture template as input (illustrated in the left part of the figure), and generates target implementations of the system (right part of the figure). SLD represents system functionalities, and an architecture template specifies target platforms. SLD is described as a set of processes running concurrently and channels representing communications among processes. Processes are written in the C language with communication APIs as interfaces to channels. A process may be implemented as either a software task on a Real-time OS (RTOS) or a hardware module with a single FSM (Finite

State Machine), depending on designer's decision on software/hardware partitioning.

Inter-process communications are represented as channels. Channels provide three kinds of communications: blocking channel, non-blocking channel, and memory channel. Blocking channels (BCs) represent a FIFO which have some buffers in it and mainly used to synchronize two processes. Non-blocking channels (NBCs) represent small data storage accessed by two or more processes. Memory channels (MEMs) are used to transfer large data between two processes. Typically, BCs and MEMs are combined and used as FIFOs with large data, which represent communications at a transaction level. NBCs are used like global variables, which are referenced by multiple processes. Channels are mapped to the memories and buses. Communication APIs used in each process description are converted to interface programs/logics to communicate with each other through channels.

4.1.2 AUTOMATIC SYNTHESIS

SystemBuilder synthesizes target implementations automatically from system described in SLD to the hardware architecture according to the mapping specification.

Processes mapped onto a processor are compiled and linked with a Real-Time OS (RTOS) for single processor system as tasks, and processes mapped on hardware modules are generated by a behavioral synthesis tool and a logic synthesis tool for a target FPGA. SystemBuilder uses TOPPERS/JSP kernel (JSP kernel) [45] for the RTOS, which is one of the most popular RTOS in Japan. eXCite[46] is used as a behavioral synthesis tool, and Altera Quartus II[47] is used as logic synthesis and place and route for a target FPGA.

In the synthesis of channels, buffers of channels are mapped on memory modules and the implementation descriptions of interfaces are generated. In brief, communications among processes on a processor are implemented as API calls of the RTOS. SystemBuilder also generates configuration files for the RTOS which register processes and channels as tasks and communication APIs, respectively. Interfaces of channels among hardware mod-

ules are converted to additional hardware modules such as fifo hardware and Block RAMs in the target FPGA. Interfaces of channels among processes on a processor and hardware modules (software-hardware communications) are realized by interrupt generation hardware and the device driver software including interrupt handler.

Figure 4.1 shows an example of synthesis result. Processes P1, P4, P5 and P7 shown in SLD are mapped on a processor CPU1, and converted into software tasks on an RTOS in a target implementation. Processes P2, P3 and P6 are mapped onto a hardware module HW1. In HW1, device registers and a bus I/F are generated for communication with CPU1. Note that mapping decision should be done by a system designer.

4.2 COSIMULATION ENVIRONMENT

SystemBuilder generates implementations of a system for not only a target FPGA but also a cosimulation environment. In prior works, a software/hardware cosimulation environment was developed [48]. The cosimulation environment enables a designer to verify functionalities of systems that consist of descriptions at multiple abstraction levels.

This section briefly describes the cosimulation environment.

4.2.1 OVERVIEW

In our past study, we developed a cosimulation environment and an RTOS model [48][49][44] for MPSoCs. The overall structure of the cosimulation environment is shown in Fig. 4.2. The cosimulation environment consists of an RTOS model, multiple hardware simulators, and a cosimulation backplane named *Device Manager (DM)*. The RTOS model supports all of the service calls which are defined by μITRON 4.0 Standard Profile [50]. ITRON is a standardized specification of RTOS for small- and mid-scale embedded systems, and is one of the most popular RTOSs in Japanese industries. The RTOS model is implemented in C,

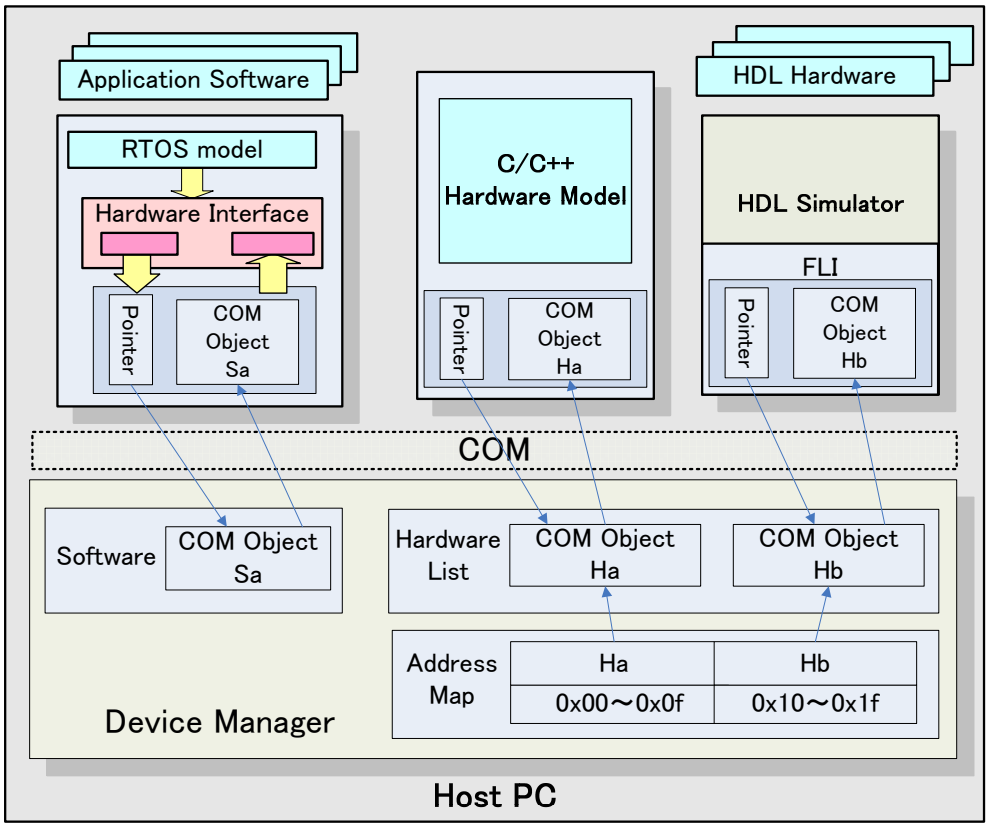


Figure 4.2: Past covalidation environment overview

so that it is directly executable on the host computer. The cosimulation environment is very flexible in that it features plug-and-play of various simulators such as HDL simulators, the SystemC simulators, functional hardware models in C/C++, and instruction-set simulators. Each simulator is executed as an application on an MS-Windows-based host computer.

In summary, the cosimulation environment developed in the past features

- native (hence fast) execution of application software,
- complete support of a standard RTOS,
- cosimulation with various hardware simulators such as HDL simulators and C/C++ functional models.

4.2.2 COMMUNICATION BETWEEN SIMULATORS

In the cosimulation environment, various simulators can communicate with each other using a flexible communication mechanism as follows[48].

Memory mapped I/O is assumed in our cosimulation environment, and unique address spaces are assigned to hardware simulators. DM manages a mapping table of the addresses and the hardware simulators. When the software needs to perform a read/write access to a hardware simulator, first the software sends an access request with an address to DM, and then DM selects a corresponding hardware simulator by looking up the address map and transfers the request to the hardware simulator.

The transfers of requests are implemented with a standard remote procedure call (RPC) on MS-Windows, named *COM*. COM is a mechanism for communications between MS-Windows applications. In order for simulators to communicate with each other, the RTOS model, hardware simulators and DM have so-called COM objects which realize the COM-based communication (shown in Fig. 4.2).

ITRON project [50] defines an API for hardware accesses. For example, application software reads from or writes to hardware devices using the following API functions.

```
x = sil_rew_mem(address);     // x=*address;
sil_wrw_mem(address, x+1);  // *address=x+1;
```

Since the cosimulation environment completely supports the API, application software does not have to be rewritten for cosimulation. For cosimulation, these APIs are translated to COM-based RPC calls to DM. For the final implementation, on the other hand, the APIs are translated to device driver software for the hardware.

4.3 MPEG-4 DECODER SYSTEM-LEVEL DESCRIPTION (SLD)

This section shows the efficiency on developing SLD through a case study on MPEG-4 decoder system design.

MPEG-4 decoder is an industry-strength application used in video cameras and cell-phones, and therefore we have taken it to be suitable as a design example.

Starting from a sequential software program, we develop system-level description with SystemBuilder by separating and refining it incrementally. We present a whole design process aiming to develop an MPEG-4 decoder system achieving 15fps (frames per second) performance, and show effectiveness of SystemBuilder on system-level design. Our case study aims to make an MPEG-4 decoder system achieve 15fps performance on a target FPGA.

We start MPEG-4 decoder system design from modifying a sequential software program into SLD. The software program of MPEG-4 decoder is selected from EEMBC benchmark suite [51]. At the end of this section, we obtain SLD where most processes can be implemented as hardware and executed concurrently in pipeline manner.

4.3.1 PRELIMINARY

In this case study, we focus on the fixed architecture, which consists of a single processor, a hardware module, a shared memory and a bus. Processes specified as software are compiled and linked with JSP kernel. Processes specified as hardware are converted to RTL (Register Transfer Level) description by a behavioral synthesis tool, as which we used a commercial tool, YXI eXCite 3.2a[46]. FPGA netlist is synthesized from RTL by Quartus II 8.0 logic synthesizer and implemented on Altera Stratix II FPGA board with a Nios II soft-core processor. The FPGA is driven at maximum speed of 100MHz and can generate variable clock frequencies for user logics with a PLL (phase locked loop). We configure the PLL to

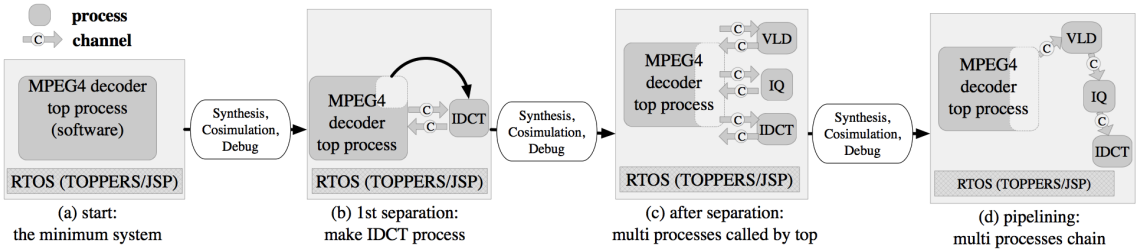


Figure 4.3: Incremental process separation.

generate maximum clock frequencies available on designed systems.

4.3.2 Initial Decision

We first decided the specifications of MPEG-4 decoder system according to input files. We have selected two files as inputs from sample files provided by EEMBC with the benchmark program: "marsface" which consists of 49 frames of 192×192 size, and "railgrind", 97 frames, 320×240 size.

Generally, MPEG-4 encoded files are sequences of GOVs (Group Of VOPs) consisting of several number of picture frames named VOP (Video Object Plane). There are three kinds of VOPs: I-, P-, and B-VOP. I-VOP is a base frame for motion compensation, and P- and B-VOP are differential frames for compaction. In detail, P-VOP consists of coded blocks and not-coded blocks. We denote them as "coded-P-VOP" and "not-coded-P-VOP" respectively. Since input files consisted of only I- and P-VOPs, we omitted other decoder features unrelated to I- and P-VOP decoding. Especially, we first focused on improving decoding performance for coded-P-VOP, which used frequently in the inputs.

4.3.3 SLD Construction

The simplest SLD is constructed of a single process and no channel (illustrated in Figure 4.3(a)). Such systems are easily made with a software program specified as a single process.

In this way, we first constructed SLD of an MPEG-4 decoder system with a single process that decodes MPEG-4 encoded files on a single processor. We call the single process as "top process" and confirmed that it is correctly executed on a Nios II processor with an RTOS.

Figure 4.3(b) illustrates the MPEG-4 decoder system after separating one process from top process. We first used GNU profiler (*gprof*) for analyzing bottlenecks. From *gprof* result, we found that *IDCT* function consumes the longest execution time on a processor and should be implemented on hardware. Thus we separated IDCT function from top process and made IDCT process. After this, as we made a new process, we generated implementations with SystemBuilder and executed the system on cosimulation platform for early debugging.

After several iterations for process separation, the system consists of several processes all connected to top process through channels (illustrated in Figure 4(c)). In the system, most processes act like software functions called by top process. Because of sequential behavior of top process, no two processes can execute concurrently and the system results in low performance. In order to improve performance, we detached each connection between top process and others, and then reconnected them to construct pipeline structure (illustrated in Figure 4.3(d)). Note that these transformations can be done by only changing locations of communication API calls in the source code of processes written in the C language.

As a result, we developed SLD which consists of ten processes: top process, header, get_mv, VLD, IQ, IDCT, MI, adder, yuv2rgb, and display. Figure 4.4 depicts the ten processes with memories for inter-process communication. Data blocks of coded-P-VOPs to be decoded are supplied by top process in succession and decoded by following processes. Processes in Figure 4.4 except for top process can be implemented as both hardware and software. In order to output not only coded-P-VOP but also I-VOP and not-coded-P-

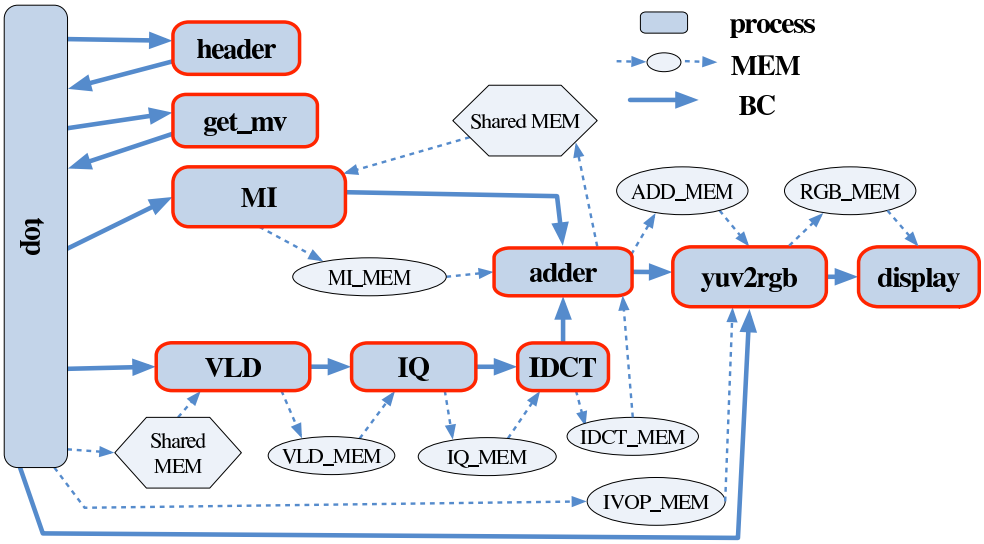


Figure 4.4: Pipelined system structure for coded-P-VOP.

VOP, the system has channels between top process and yuv2rgb process (illustrated as IVOP_MEM and a solid arrow in Figure 4.4). I- and not-coded-P-VOP are decoded by top process, and transferred to yuv2rgb process through the channels.

4.4 ADVANTAGES AND PROBLEMS

This section brings advantages and subjects of SystemBuilder by referring sections described above.

We found four advantages through this case study as follows.

(1) We could reuse a software program of MPEG-4 decoder for initial design, because SystemBuilder takes SLD written in the C language, which is one of the most popular language in embedded software design.

(2) Abstract representation of inter-process communications helped us transform SLD construction, i.e. process separation and pipelining (shown in Section III).

(3) Automatic communication synthesis by SystemBuilder reduced overall design time, although SLD should be modified manually for transformation,

(4) Code refinement is generally error prone, however, cosimulation support of SystemBuilder enabled early verification.

Problems were also found in this case study.

(1) SystemBuilder cannot explore multiprocessor system design. Recently LSIs which have multiple processors, called MPSoCs, have been developed and used. In order to utilize MPSoCs, automatic synthesis tools should support exploration of multiprocessor systems.

(2) Verification using cosimulation environment was slow for an MPEG-4 decoder system. Since recent embedded systems have been increasing their complexity and size like the MPEG-4 decoder system, the slow execution of cycle-level simulation tools are not sufficient for verification of such systems. Some techniques for accelerate verification is needed.

(3) Analysis tools for FPGA-based prototypes of systems. Although FPGA-based prototypes are fast and accurate for evaluation of mappings, it is difficult to figure out the reason of the result since the internal behavior such as activation/wait timings of processes could not be observed with FPGA-based prototypes.

(4) automatic generation of design candidates according to mapping specification takes long time. Although the manual modification of description for implementation according to mapping specifications are removed by SystemBuilder, even synthesis time by behavioral and logic synthesis tools are long. Since these tools takes several minutes or hours, exploration of millions of design candidates will end up with taking days or months.

4.5 CONCLUSIONS

This chapter presented an overview of SystemBuilder, a prior work of SystemBuilder-MP. SystemBuilder supports design space exploration at system level by automatically synthesizing system implementation from abstract functional description of systems according to mapping specifications. With the automatic synthesis capability, system designers need not modify implementation descriptions of a system manually for mapping exploration and can explore design candidates easily.

We also demonstrated a case study on MPEG-4 decoder system design with SystemBuilder. The MPEG-4 decoder system description was developed by converting a sequential software program. Until the completion of system design, a number of design-implement-evaluate steps were iteratively performed to construct system-level description and to refine it. Finally, we developed functional description of an MPEG-4 decoder system with pipelined parallelism. Most of processes in the MPEG-4 decoder description can be implemented as both software and hardware. Through the case study, we found the easiness of design space exploration with SystemBuilder and problems to solve of SystemBuilder and its methodology.

CHAPTER 5

SYSTEMBUILDER-MP

5.1 INTRODUCTION

As the complexity of embedded systems grows to the extent of MPSoCs (Multi-Processor System-on-a-Chip), design space exploration at a system level plays a more important role than before. For example, multimedia systems such as cell phones, video cameras and TV recorders utilize multi-processors and dedicated accelerators in order to achieve high performance for real-time media processing.

There are some examples of systems which have both multiple processors and dedicated hardware modules. Toshiba proposed MeP (Media embedded Processor) [52] and Renesas Electronics provides LSIs for cell phones [53]. These LSIs have not only multiple processors but also dedicated hardware for playing and recording multimedia such as movies and audio. The needs for these LSIs are clearly showing that importance of both multiple processors and dedicated hardware modules for embedded systems which requires both high performance and real-time processing. The target of our system-level design tool is such LSIs for multimedia. In such systems functionalities should be distributed properly on processors and hardware modules, and run in parallel. System designers should explore

the design space considering the parallelism of the system.

We have developed integrated system-level design tool set, named SystemBuilder-MP. SystemBuilder-MP provides abstract functional programming model and automatic synthesis capability for FPGA-based multiprocessor system prototyping. With automatic synthesis capabilities, system designers easily explore design space with multiprocessors and dedicated hardware modules. Especially, SystemBuilder-MP have abstract communication representation at system-level and automatic synthesis of implementation focusing on multiprocessor systems. This chapter presents overall system design methodology and individual technologies of SystemBuilder.

Recently FPGAs have been becoming more and more important for embedded systems. Traditionally FPGAs are used only for prototyping in a design phase. However, the growth of FPGAs and needs for configurability made FPGAs product-level device. This is why SystemBuilder-MP synthesise FPGA-based prototypes for evaluation of systems. The synthesis results can be used as both prototypes and products. In the design of ASICs (Application Specific Integrated Circuits), A FPGA-based prototype as a result of SystemBuilder-MP may be used as a prototype used for verification and evaluation of the system. On the other hand, FPGAs have been used in embedded systems as not a prototype but a final product.

The rest of this chapter is organized as follows. First, Section 5.2 defines the design target of SystemBuilder-MP. Next, Section 5.3 shows an overview of the design flow achieved by SystemBuilder-MP. Section 5.4, 5.5 explain details of inputs and automatic synthesis functionalities, respectively. Then Section 5.6 shows the effectiveness of SystemBuilder-MP on MPEG-4 decoder case study. Finally Section 5.7 concludes this chapter with a summary.

5.2 DESIGN TARGETS

SystemBuilder currently uses Altera's tools as its back-end for logic synthesis and place-and-route, and therefore, the target architecture of SystemBuilder is restricted to one supported by Altera's tools. Specifically, Nios II soft-core processors with Avalon interconnects are supported by SystemBuilder at present.

There is no restriction on the numbers of processors, hardware accelerator modules, memory modules and buses, as many as the FPGA device allows. The numbers of these modules and the interconnection between them are defined by designers in an input file of SystemBuilder (described as "architecture template" in Sec. 5.4.3).

Mapping of processes onto processors is statically determined at a design phase and is not changed at runtime. SystemBuilder also assumes that a single address space is shared by all the modules. The two assumptions are realistic and very popular in many embedded systems in order to meet real-time requirements[54].

By default, SystemBuilder-MP generates a single master interface port (MIF) and a single slave interface port (SIF) for a dedicated hardware module. MIFs are used by processes on the hardware module to outside memory modules and SIFs are used by processes on processors in order to access memories inside the hardware module.

It should be noted that, although SystemBuilder at present supports only Altera's FPGAs and their associated architectures, SystemBuilder can potentially support other devices and architectures. In actual, an earlier version of SystemBuilder supported Xilinx's architecture with Microblaze soft-core processors and the OPB bus[55].

5.3 DESIGN FLOW WITH SYSTEMBUILDER-MP

This section explains the design flow achieved with SystemBuilder-MP. Fig. 5.1 illustrates overview of the design flow, and followings explain the design flow according to the figure.

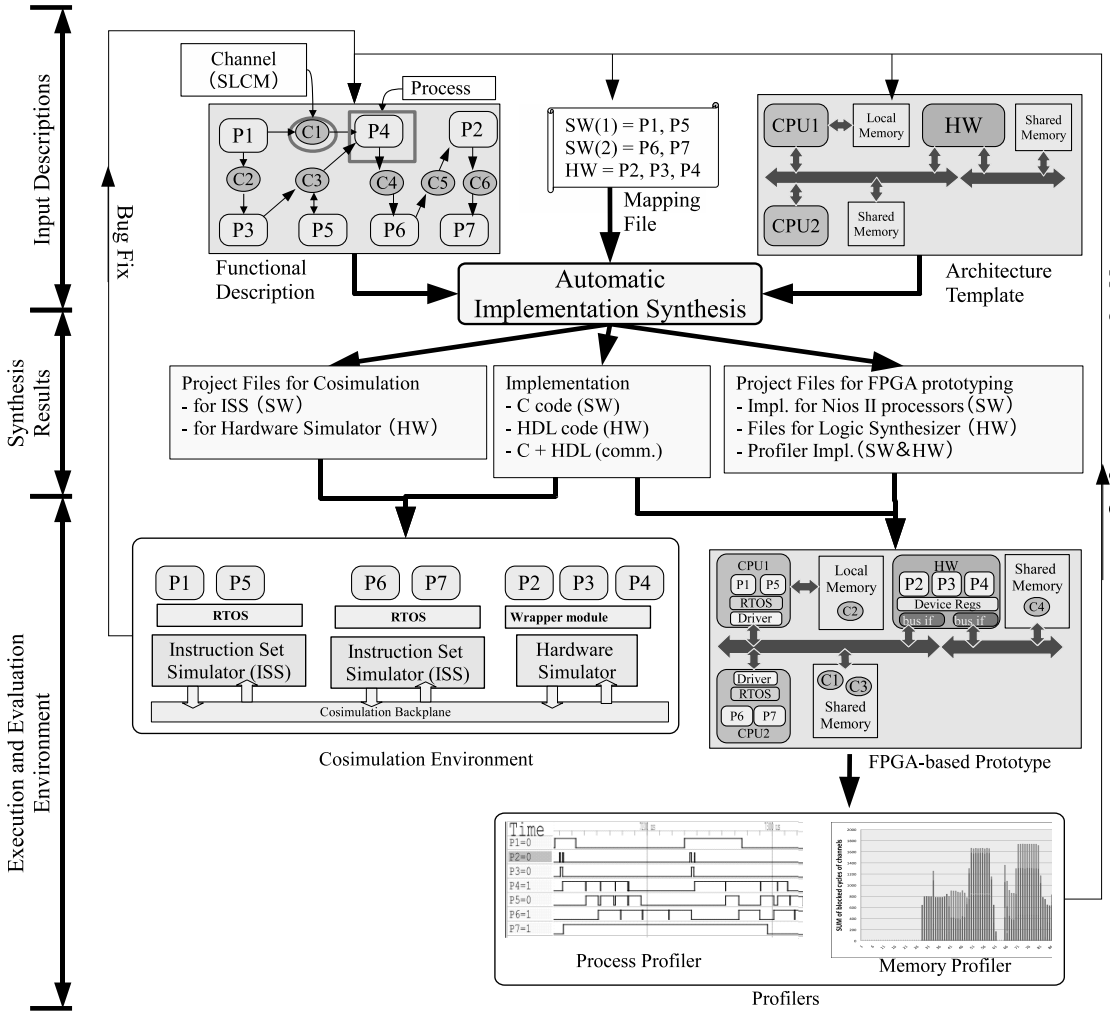


Figure 5.1: Design Flow of SystemBuilder-MP.

5.4 INPUT DESCRIPTIONS

First, a system designer develops a “functional description” to capture functionalities of the target system. The designer also specifies hardware architecture in an “architecture template” and mapping of the processes and the channels onto the hardware architecture in a “mapping specification”.

The functional description consists of “processes” and “channels”. A process and a channel represent a computation component and inter-process communication at a high

abstraction level, respectively.

5.4.1 PROCESSES

Processes represents functionality of a system. A process is a sequence of computation and a unit for mapping exploration. A process may be mapped onto one of processors and hardware modules. Processes communicates each other using channel.

Processes are described in the C language. In order to use channels in the C description, SystemBuilder-MP provides channel APIs.

5.4.2 CHANNELS

Channels are abstract communication method for processes. The definition of channels only defines their behaviors. In the synthesis phase, their implementation (software or hardware) and implementation detail (how to implement as software or hardware) are decided.

SystemBuilder-MP defines five channels: blocking, non-blocking, memory, exclusive control and ring buffer control.

Blocking channels (BCs), non-blocking channels (NBCs) and memory channels (MEMs) are same as described in 4.1.1. Here, we describe two newly developed channels: exclusive control and ring buffer control.

Exclusive control channel (EXC) represents a lock. In the multiprocessor system, lock is one of most important mechanism for using shared resources. EXCs provide locks among processes. Fig. 5.2 shows an usage example of EXC. In the figure, an EXC named `EXC1` is used. At first line, a call of "`EXC1_LOCK()`" acquires a lock of EXC1. With the call, the process waits until it obtain the lock. After obtaining the lock, the process executes computation using some shared resources (critical section). Finally the process releases the

```
EXC1_LOCK(); // lock
/* critical section */
EXC1_RELEASE(); // unlock
```

Figure 5.2: An example of lock primitive.

```
//get 10 empty buffers
FIFO_RBC_ACQUIRE_ROOM(10);
//write contents of array b[10] to empty buffers
FIFO_RBC_STORE(b);
//release 10 buffers with written data
FIFO_RBC_RELEASE_DATA();
```

Figure 5.3: An example of ring buffer primitive.

lock by calling "`EXC1_RELEASE()`".

Ring buffer control channel (RBC) is a memory manager. By using an RBC with a MEM, designers can easily construct a FIFO with large data (ring buffer, in other words). RBCs manage memory regions and order of usage of the regions. Fig. 5.3 shows an usage example of an RBC. In the figure, a process uses "`FIFO_RBC`". The process first try to obtain 10 empty buffers by calling "`FIFO_RBC_ACQUIRE_ROOM(10)`". After obtaining the buffers, it computes some data and write 10 data in `b[10]` to the buffer. Finally it release the buffer with the written data by calling "`FIFO_RBC_RELEASE_DATA()`". The written data may be read by other processes.

All APIs provided by channels are shown in Appendix A.

5.4.3 ARCHITECTURE TEMPLATE

The architecture template describes hardware architecture on which systems are implemented. For design space exploration, followings can be described in architecture template.

- the number of processors

- the existence of a hardware module
- the number and kinds of memory modules
- the number of bus interfaces on a hardware module.

To be concrete, SystemBuilder-MP takes architecture template described in a description format supported by Altera SOPCBuilder. Interconnection among hardware modules such as processors and dedicated hardware modules and address maps are set in SOPCBuilder.

5.4.4 MAPPING SPECIFICATION

In the mapping specification, designers can specify mapping of processes and channels onto hardware architecture specified in architecture template.

An example of the mapping specification is shown in Fig. 5.4. Lines start with character "#" are comments and have no meaning as input for SystemBuilder-MP. Mapping of processes are shown in from line 2 to line 4. "SW(1)", "SW(2)" mean processors. In the example, process P1 and P5 are mapped onto a processor and processes P6 and P7 are mapped onto another processor. Rest of processes P2, P3 and P4 are mapped onto a dedicated hardware module (HW). System designers can explore mapping only by changing these lines.

Mapping of not only processes but also memory channels can also be specified. If designers specify MIFs and SIFs in mapping specification like line 6 to line 9, SystemBuilder-MP generates more than one interface ports and maps memory channels on them. The use of multiple interface ports is effective to avoid access conflicts on the ports. In the example, memory channel `c3_SMEM1` is mapped onto MIF(1) and memory channel `c4_SMEM2` is mapped onto MIF(2). With this mapping, accesses for `c3_SMEM1` and `c4_SMEM2`

```
1: # Mapping of processes to processors/hardware module
2: SW(1)    = P1, P5
3: SW(2)    = P6, P7
4: HW       = P2, P3, P4
5: # Mapping of channels to bus interfaces
6: MIF(1)   = c3_SMEM1              // Master interface 1
7: MIF(2)   = c4_SMEM2              // Master interface 2
8: SIF(1)   = c1_P1toP2             // Slave interface 1
9: SIF(2)   = c5_P6toP4             // Slave interface 2
```

Figure 5.4: An example of a mapping file.

do not conflict at master interface of dedicated hardware module. Similarly, accesses for `c1_P1toP2` and `c5_P6toP4` do not conflict at slave interface.

5.5 AUTOMATIC SYNTHESIS

SystemBuilder-MP automatically synthesizes implementation descriptions of interconnections among processes from channel (hereafter, this synthesis functionality is called as "communication synthesis"). The synthesized communication descriptions are in the C language and VHDL, depending on mapping of the processes and channel.

In the communication synthesis, buffers of channels are mapped on memory modules and the implementation descriptions of interfaces are synthesized. In brief, communications among processes on processors are implemented as API calls of an RTOS. The RTOS handles both intra-processor and inter-processor communication.

5.5.1 SYNTHESIS OF PROCESSES

Processes mapped onto processors are implemented as a task of RTOS on their processors by automatic synthesis. For multiprocessor design, SystemBuilder-MP uses TOP-

```
local_class PE1{
    CRE_TSK(P1_TASK, {TA_HLNG|TA_ACT, (VP_INT)1,
      P1_task, 10, 4096, NULL});
    CRE_TSK(P5_TASK, {TA_HLNG|TA_ACT, (VP_INT)1,
      P5_task, 10, 4096, NULL});
    ...}
local_class PE2{
    CRE_TSK(P6_TASK, {TA_HLNG|TA_ACT, (VP_INT)1,
      vld_task,10, 4096, NULL});
    CRE_TSK(P7_TASK, {TA_HLNG|TA_ACT, (VP_INT)1,
      vld_task,10, 4096, NULL});
    ...}
```

Figure 5.5: An example of a CFG file of TOPPERS/FDMP kernel.

PERS/FDMP kernel (FDMP kernel) [55] as RTOS. For single processor design, SystemBuilder-MP uses JSP kernel in the same way as SystemBuilder.

SystemBuilder-MP automatically generates setting files of RTOSs (CFG files) which specifies informations such as task creation. Fig. 5.5 shows a CFG file generated according to mapping specification in Fig. 5.4. In the CFG file, a group named “class P1” have two tasks P1 and P5. In TOPPRES/FDMP kernel, a class corresponds to a processor. Moreover, SystemBuilder-MP generates Makefiles and liker scripts which are used to compile software.

Processes mapped onto hardware are converted to hardware by the behavioral synthesis tool eXCite [46] in the same way as SystemBuilder. eXCite converts behavioral description in the ANSI C language into hardware description language (HDL) at register transfer level (RTL) such as VHDL and Verilog. Altera Quartus II[47] is used for logic synthesis and place and route to configure a target FPGA.

5.5.2 SYNTHESIS OF CHANNELS

Implementation of channels is decided by mapping of processes. There are three kinds of communication from a view point of implementation: software-software communication, hardware-hardware one and software-hardware one. Hereafter, we describe these three kinds of communication.

SOFTWARE-SOFTWARE COMMUNICATION

In detail, software-software communication is divided into intra-processor communication and inter-processor one. Designers, however, need not to consider such difference since SystemBuilder-MP automatically generates appropriate implementation.

BCs, EXCs and RBCs are implemented by using APIs of FDMP kernel. FDMP kernel appropriately handles both intra- and inter-processor communication. Concretely, BCs are implemented using queue APIs. EXCs are implemented using semaphore APIs. RBCs are impelemented using semaphore APIs and memory management C programs.

On the other hand, NBCs and MEMs are handled by SystemBuilder-MP at synthesis phase. If a channel is intra-processor one, SystemBuilder-MP generates a processor-local memory region. If a channel is inter-processor one, SystemBuilder-MP generates a global memory region.

HARDWARE-HARDWARE COMMUNICATION

Channels among hardware processes are implemented as hardware circuitry inside a dedicated hardware module. An example is shown in Fig. 5.6 (described later) as “FIFO2”.

For BCs, EXCs and RBCs, FIFO hardware, lock hardware and memory management hardware are generated, respectively. NBCs and MEMs are implemented as registers and memory modules in the dedicated hardware module, respectively.

SOFTWARE-HARDWARE COMMUNICATION

Channels among software processes and hardware processes are implemented as hardware circuitry inside a dedicated hardware module with interfaces (SIFs) which can be accessed by processors. SystemBuilder-MP also generates device driver description for software.

Similar to hardware-hardware communication implementaion, FIFO hardware, lock hardware and memory management hardware are generated for BCs, EXCs and RBCs, respectively. Since these hardware may block execution of software processes, SystemBuilder-MP also generates interruption circuitry for notification of the end of block to processors. Interruption circuitry is connected only to corresponding processors. Moreover, interrupt handler description for software is generated.

NBCs and MEMs are implemented as registers and memory modules in the dedicated hardware module, respectively. For corresponding software processes, description for accessing them through memory-mapped I/O are generated.

5.5.3 AUTOMATIC GENERATION OF MULTIPLE BUS INTERFACES

SystemBuilder-MP generates implementation of MIFs and SIFs according to mapping specification (an example is shown in Fig. 5.4).

Fig. 5.6 illustrates hardware architecture synthesised from mapping specification shown in Fig. 5.4. In the example, processes P2, P3 and P4 are implemented in a hardware module and implementation of channels corresponding to them (FIFOs and Memory) are in the hardware module. FIFO hardware and Memory modules corresponding to P2 and P3 are accessed by processes on processsor1 through SIF1, and P4 communicates with processes on processor2 through SIF2. If there are no specification about SIFs, all accesses P3 to SMEM1 and P4 to SMEM2 use a single MIF (MIF1), and may cause access conflicts on system execution.

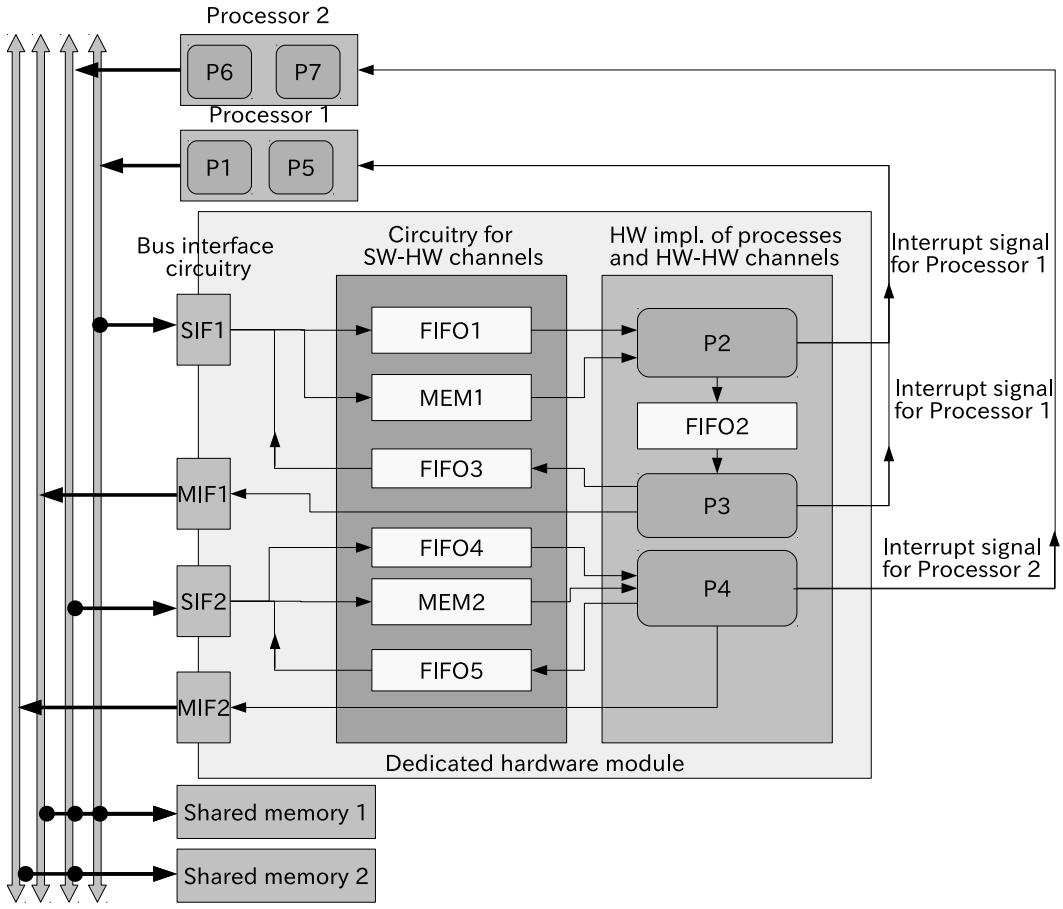


Figure 5.6: An example of a generated system with multiple bus interfaces.

There are two memory modules (“SMEM1” and “SMEM2”) in the example. P3 accesses SMEM1 through MIF1 and P4 accesses SMEM2 through MIF2. If there are no specification about MIFs, both two accesses (P3 to SMEM1 and P4 to SMEM2) use a single MIF (MIF1), and may cause access conflicts on system execution.

5.6 EVALUATION OF MULTIPROCESSOR DESIGN BY MPEG-4 DECODER CASE STUDY

This section evaluates effect of SystemBuilder-MP for multiprocessor system design by case study on MPEG-4 decoder system design. The evaluation of multiple bus interface generation capability is conducted in 7.3.2 with evalution of system-level profilers.

5.6.1 EXPLORATION TIME

We designed an mpeg4 decoder system and evaluated the effectiveness of SystemBuilder-MP.

Fig. 5.7 shows the functional description of the mpeg4 decoder system. It consists of 10 processes and many channels (shown partly in the figure). There are some design choices on the mpeg4 decoder. All processes except for "top" process can be mapped onto either processors or hardware modules. The top process is mapped only on a processor.

In this case study, we explored mapping of the mpeg4 decoder onto a multiprocessor system whose number of processors (1 to 4) and memory modules can be configured. Each process of the mpeg4 decoder was mapped on the processors or on the hardware modules. Each channel was explored mapping of the buffers onto two different memory modules, on-chip memories and off-chip SDRAM.

Target FPGA was Altera Stratix II FPGA with multiple Nios II soft-core processors running at 50MHz.

We explored 24 mappings shown in Table 5.1. Fig. 5.8 illustrates the exploration results of the 24 mappings about decode time and area trade-off. Decode time was measured by decoding time of an mpeg4 video, and area was measured by the number of ALUTs of the target FPGA.

The 24 results plotted in Fig. 5.8 were obtained only in about five hours with an exist-

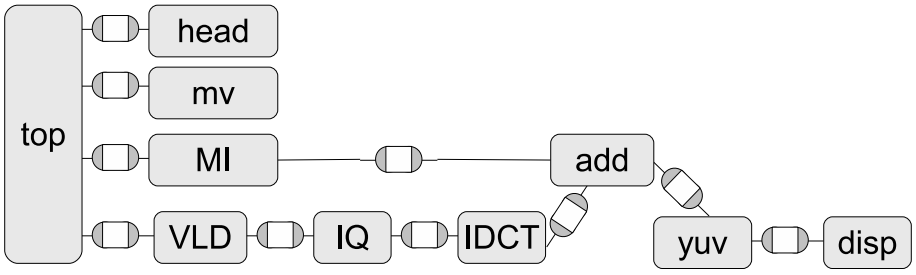


Figure 5.7: Functional description of an mpeg4 decoder.

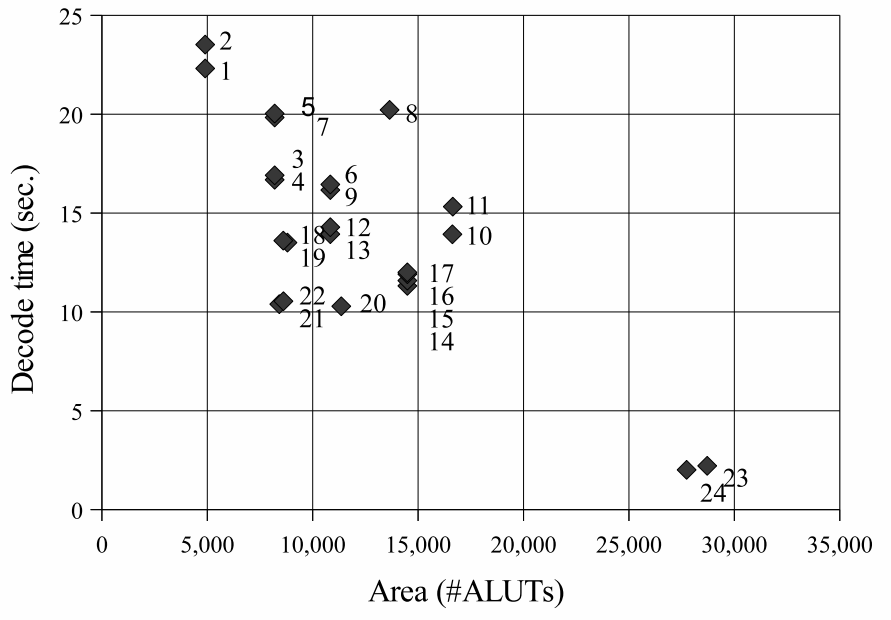


Figure 5.8: Exploration results of an MPEG4 decoder system on multiprocessor architecture with dedicated hardware modules.

ing functional description of the mpeg4 decoder. The most time was spent by a behavioral synthesis tool and logic synthesis tool. The communication synthesis of Advanced System-Builder was completed in several seconds, and execution of FPGA-based prototypes was performed in several minutes.

Table 5.1: Explored mapping of the MPEG4 decoder system.

	top	head	mv	VLD	IQ	IDCT	MI	add	yuv	disp	MEM
1	1	1	1	1	1	1	1	1	1	1	onchip
2	1	1	1	1	1	1	1	1	1	1	SDRAM
3	1	1	1	1	1	1	2	2	2	2	onchip
4	1	1	1	1	1	1	2	2	2	2	SDRAM
5	1	1	1	2	2	2	1	1	1	1	onchip
6	1	1	1	2	2	2	3	3	1	1	onchip
7	1	1	1	2	2	2	1	1	1	1	SDRAM
8	1	HW	HW	2	2	2	1	1	1	1	SDRAM
9	1	1	1	2	2	2	3	3	1	1	SDRAM
10	1	HW	HW	2	2	2	3	3	1	1	SDRAM
11	1	1	1	2	2	2	HW	HW	1	1	SDRAM
12	1	1	1	2	2	2	1	1	3	1	onchip
13	1	1	1	2	2	2	1	1	3	1	SDRAM
14	1	1	1	2	2	2	3	3	4	1	onchip
15	1	1	1	2	2	2	3	3	4	1	SDRAM
16	1	1	1	2	2	2	3	3	4	2	onchip
17	1	1	1	2	2	2	3	3	4	2	SDRAM
18	1	1	1	2	2	2	1	1	HW	1	onchip
19	1	1	1	2	2	2	1	1	HW	1	SDRAM
20	1	1	1	2	2	2	3	3	HW	1	SDRAM
21	1	1	1	2	2	2	1	1	HW	HW	onchip
22	1	1	1	2	2	2	1	1	HW	HW	SDRAM
23	1	HW	HW	HW	HW	HW	HW	HW	HW	HW	onchip
24	1	HW	HW	HW	HW	HW	HW	HW	HW	HW	SDRAM

5.6.2 EASINESS OF MULTIPROCESSOR SYSTEM DESIGN

This section shows the easiness of multiprocessor design.

Fig. 5.6.2 shows functional structure of MPEG-4 decoder. Rectangles represent processes and edges between processes represent data dependencies between them. The MPEG-4 decoder have 12 processes. Unlike the structure shown in Fig. 4.4, MPEG-4 decoder used in this section have two “interpolate” process for higher parallelism.

This case-study shows easiness of exploration of multiprocessor design.

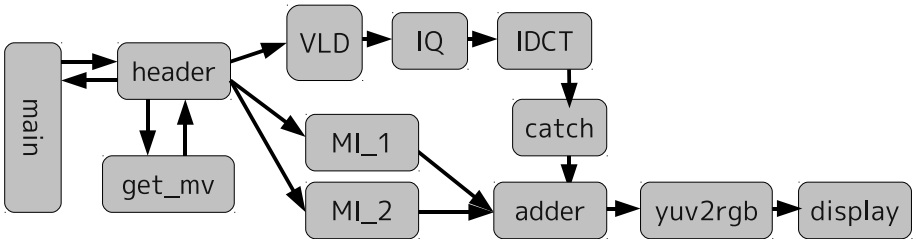


Figure 5.9: The functional structure of the MPEG4 decoder system.

mapping	main	header	get_mv	VLD	IQ	IDCT	catch	MI_1	MI_2	adder	yuv2rgb	display	exec. time
1-a	1	1	1	1	1	1	1	1	1	1	1	1	25.77 sec.

(a) with a single processor

mapping	main	header	get_mv	VLD	IQ	IDCT	catch	MI_1	MI_2	adder	yuv2rgb	display	exec. time
2-a	1	1	1	1	1	1	1	2	2	1	1	1	25.34 sec.
2-b	1	1	1	1	1	1	1	2	2	2	2	1	19.53 sec.
2-c	1	1	1	1	1	1	1	2	2	2	2	2	20.83 sec.
2-d	1	1	1	1	1	1	1	1	1	1	2	2	19.28 sec.
2-e	1	1	1	1	1	2	2	1	1	1	2	2	16.46 sec.

(b) with two processors

mapping	main	header	get_mv	VLD	IQ	IDCT	catch	MI_1	MI_2	adder	yuv2rgb	display	exec. time
3-a	1	2	2	2	2	2	2	3	3	3	1	1	19.59 sec.
3-b	1	1	1	2	2	2	2	3	1	3	1	1	22.31 sec.
3-c	1	1	1	2	2	2	2	2	3	3	3	3	22.35 sec.
3-d	1	1	1	1	2	1	2	3	1	3	2	3	17.09 sec.
3-e	1	1	1	1	2	3	2	3	1	3	2	3	16.24 sec.

(c) with three processors

mapping	main	header	get_mv	VLD	IQ	IDCT	catch	MI_1	MI_2	adder	yuv2rgb	display	exec. time
4-a	1	2	2	2	2	2	2	1	3	3	4	3	22.52 sec.
4-b	1	1	1	2	2	2	2	3	1	3	4	4	21.47 sec.
4-c	1	1	1	2	2	2	2	3	3	3	4	2	21.12 sec.
4-d	1	1	1	1	2	3	2	3	1	4	2	3	19.53 sec.
4-e	1	1	1	1	2	3	2	3	1	3	4	3	19.59 sec.

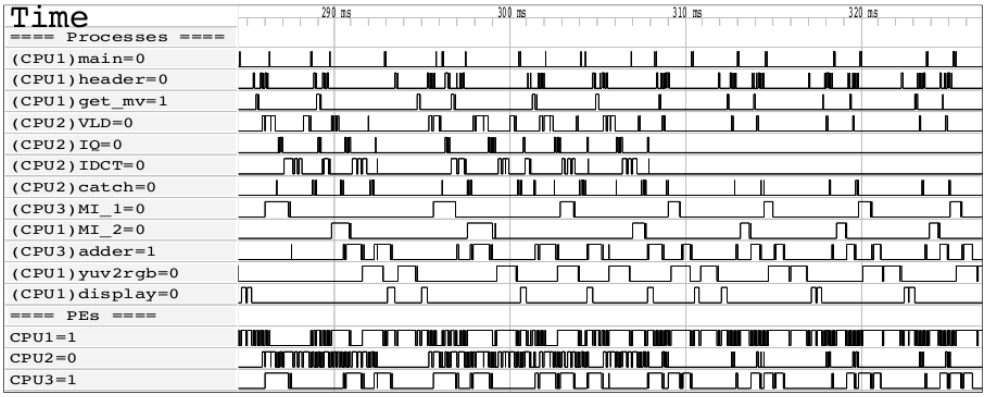
(d) with four processors

Figure 5.10: An exploration result of multiprocessor mapping.

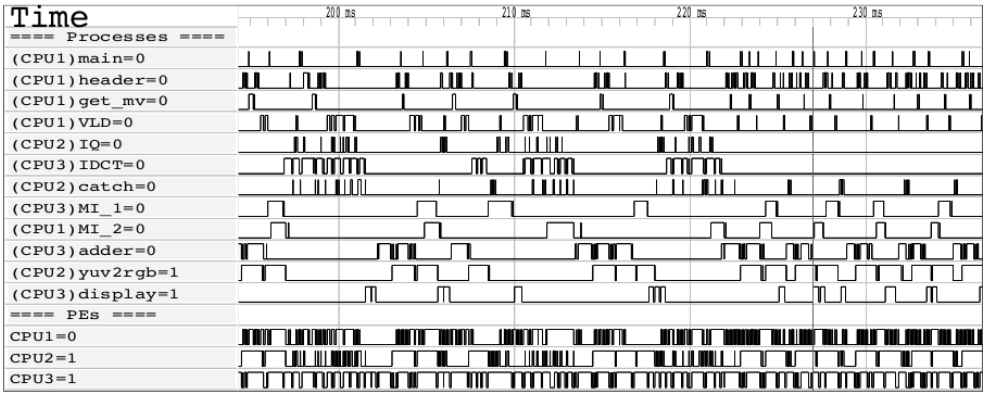
We explored mapping of the 12 processes onto hardware architecture with up to four processors. In this case-study, we did not mapped processes onto hardware.

Input for the MPEG-4 decoder is an 192×192 sized MPEG-4 file with 49 frames. The objective of exploration is to find mappings which decodes faster than others on each number of processors.

Exploration results are shown in Fig. 5.6.2. (a), (b), (c) and (d) in Fig .5.6.2 show mapping of processes and spent time to decode (exec. time) on architecture with one, two, three



(a) Process profile of the mapping 3-b



(b) Process profile of the mapping 3-e

Figure 5.11: A comparison of process profiles of mapping 3-b and 3-e.

and four processors, respectively. In the figures, the numbers in the cell means processor id numbers. For instance, the mapping 2-a in Fig. 5.10(b) is a mapping where processes `MI_1` and `MI_2` are mapped onto processor 2 and others are mapped onto processor 1.

We could find that mapping 3-e is fastest in mappings in Fig. 5.6.2. Intuitively, it is not obvious that the mapping 3-e is the fastest since `IDCT` and `MI_1` are mapped onto the same processor and their parallelism are not used in the mapping. For example, the mapping 3-b can be thought to be faster than the mapping 3-e since `IDCT` and `MI_1` are mapped onto different processors in 3-b and their parallel execution may contribute fast decoding. Therefore we analyzed the behavior of processes using the process profiler.

Fig. 5.6.2 shows comparison of process profiler results of mappings 3-b (5.11(a)) and 3-e (5.11(b)). In the results, we show waveforms of activation/wait timings of not only pro-

cesses but also processors (“CPU1”, “CPU2”, “CPU3” in the figures 5.11(a) and 5.11(b)). Actually, a waveform of a processor is union of waveforms of processes on the processor.

By comparing the figures 5.11(a) and 5.11(b), we could find difference of behavior of processes on the right side of profiles. In the mapping 3-e, the `main` process was activated more frequently than that in the mapping 3-b. This was because the activations of `main` process were interfered by other processes on the same processor such as the `yuv2rgb` process. Since the MPEG-4 decoder executes in pipelined manner, the delay of activation of topmost process results in delay of all other processes. In contrast, in the mapping 3-e, processes in former part of pipeline such as `main`, `header` and `get_mv` were not interfered by processes in latter part and therefore the mapping 3-e achieved better execution time by smooth pipelined parallel execution.

We also shows the goodness of the mapping 3-e from another view point, CPU utilization. From waveforms of processors in the figures 5.11(a) and 5.11(b), we could see that utilization of CPU2 and CPU3 in the mapping 3-b are lower than those in the mapping 3-e. In concrete, utilization of processors on the mapping 3-b were 64%, 13% and 31% for CPU1, CPU2 and CPU3, respectively (36% in average). In contrast, utilization of processors on the mapping 3-e were 38%, 61% and 71%, respectively (57% in average). This shows higher parallelism of the mapping 3-e clearly.

We think the analysis results shown in above is a good example that shows design exploration using FPGA-based prototypes contributed to find good mapping which is not intuitive.

Decoding time on architecture with four processors were slower than those with three processors. The reason is thought to be delay by the conflicts on accesses to a shared memory.

As a result of this exploration, we could find fast mapping with fewer number of processors.

The time to explore 16 mappings in Fig. 5.6.2 was only about two hours in total. In detail, we took an hour to synthesize four FPGA configurations using a logic synthesis tool. After that, 10 minutes were spent to synthesize implementation of 16 mappings using SystemBuilder-MP. Compilation of software of 16 mapping took about 30 minutes. Finally, downloading hardware and software images into FPGA and system execution of 16 mappings took 20 minutes in total.

In the exploration time detail, the effect of SystemBuilder-MP were seen in implementation synthesis and software compile. If there did not exist SystemBuilder-MP, implementation of 16 mappings must not be so easy since manual implementation of these mappings needs preparation of many files. For instance, if designers want to change mapping of processes on an architecture with a certain number of processors, they might need to change CFG files of RTOS (shown in Fig. 5.5) and Makefile for compilation. Moreover, if they want to change the number of processors, they need to change not only CFG files but also linker scripts. If the number of processors is changed from 1 to 2 or more, and vice versa, RTOS should also be changed from the one for single processor (TOPPERS/JSP kernel in our case) to the one for multiprocessor (TOPPPERS/FDMP kernel in our case). By automating these changes using SystemBuilder-MP, we achieved efficient exploration.

In this case-study, the number of lines of files which automatically generated by SystemBuilder-MP are about 3560 in average for mappings. This includes C codes, CFG files for RTOS, Makefiles and linker scripts. In these files, hundreds of lines were different by mappings. For instance, the number of lines which were different between files of mappings 4-a and 4-d was about 550. It means that if SystemBuilder-MP did not exist, designers should change manually about 550 lines distributing several files. It should be noted that this number is obtained by comparison of mappings on same architecture (with four processors). If the number of processors was also changed, designers should change more lines.

The manual change of hundreds of lines may lead to implementation of bugs by hu-

man errors. Bug implementation may result in delay of exploration for debug and re-compilation of software. In contrast, this case-study needed only a few manual changing of mapping specification and do not suffered from a bug. Because of automatic synthesis of SystemBuilder-MP, we could concentrate exploration and analysis of mappings.

From results above, efficiency of design space exploration with SystemBuilder-MP were shown.

5.7 CONCLUSIONS

This chapter presented overall system-level design methodology achieved by our tool, named SystemBuilder-MP. System designer can describe the system and explore design candidates in a short turn-around-time with automatic synthesis capabilities and verification/evaluation support tools provided by SystemBuilder-MP. A case study of mpeg4 decoder system design demonstrated efficiency of SystemBuilder-MP by exploring 24 design candidates which differ in the number processors and the memory modules, and mapping of processes onto processors and hardware modules.

CHAPTER 6

COVALIDATION ENVIRONMENT

6.1 INTRODUCTION

Software and hardware for embedded systems have been increasing their size and complexity, while the time-to-market pressure has also been increased. In order to satisfy both of the requirements, fast software/hardware covalidation is one of the key technologies.

In typical embedded real-time systems, software consists of application tasks and an RTOS, and therefore, RTOS should be incorporated in the software/hardware covalidation flow in order to verify the overall system functionality. In the past, several researchers developed simulation models of RTOSs to be used in their hardware/software cosimulation frameworks [23, 56, 57]. They assume that all of the system components (including software components and hardware ones) are written in a single system-level description language (SLDL) such as SystemC and SpecC. Although their approaches lead to fast cosimulation, one of their serious drawbacks is that hardware components in the SLDL are often just simulation models whose detailed functionality might be different from that of the final implementation descriptions in HDL.

In our prior work, an RTOS simulation model and a multilingual cosimulation platform

on which HDL simulators can be executed was developed (described in 4.2). While the RTOS model can be executed natively (hence fast) on a host computer, HDL simulators are inevitably slow. Such cosimulation is appropriate for hardware debugging, but inappropriate for functional verification of embedded software.

In this work, we have developed a software/hardware covalidation environment to be used for embedded software verification. In the covalidation environment, embedded software is executed directly on the host PC while hardware is executed on an FPGA. This work solves the two problems of the past approaches at the same time. Since final HDL designs (not simulation models) can be used for the software/hardware covalidation, unexpected inconsistency between software and hardware can be avoided. In addition, our covalidation environment brings the significant speedup compared with traditional cosimulation using HDL simulators. It should be noted that our covalidation environment presented in this chapter is complementary to the past cosimulation environments. This work provides yet another covalidation solution to embedded software designers.

The covalidation environment presented in this chapter was built upon our prior work on cosimulation described in 4.2.2.

This chapter is organized as follows. Section 6.2 describes the covalidation environment with an RTOS model and FPGA. A case study with an MPEG4 decoder system is presented in Section 6.3. Finally, Section 6.4 concludes this chapter with a summary.

6.2 COVALIDATION WITH RTOS MODEL AND FPGA

As shown in the previous section, the hardware/software cosimulation environment which we developed in the past is very flexible. Specifically, it is useful for hardware debugging since software can serve as a fast, interactive testbench. For software debugging, however, the cosimulation environment is less efficient since HDL simulators are inevitably slow. In

order to improve the execution speed, in this work, we have extended the environment to be able to connect to an FPGA.

We have developed two types of FPGA connection. One type uses RPC-based communication to make the most of communication flexibility of the environment (described in 4.2). Because of the flexibility of the RPC-based communication, software and hardware do not have to be modified from HDL simulation to FPGA execution. Another type uses direct communication from the RTOS model to FPGA. The latter type enables faster covalidation than the former type because of less communication overhead. Application software do not have to be modified regardless of the type of FPGA connection because both communication types are implemented under the hardware interface API of the ITRON standard.

With the former type which uses RPC-based communication, software can communicate with FPGA in the same way as when connected with an HDL simulator, because FPGA is connected to DM through *FPGA process* provided by our covalidation environment (illustrated in Fig. 6.1). The FPGA process is an MS-Windows process which intermediates the communication between DM and the FPGA. In brief, the FPGA with the FPGA process is equivalent to the HDL simulator from a view point of software. The FPGA process has a COM object to communicates with DM, and through the DM, the software performs read/write to the FPGA. When the software needs to communicate with the hardware implemented on the FPGA, first the software sends a request to DM, next DM dispatches the request to the FPGA process, and then the FPGA process actually reads from or writes to the FPGA. For the sake of flexibility, this type of connection causes a large overhead caused by RPC communication which costs over 0.1 milliseconds per communication.

With the latter type, the FPGA communicates directly with the RTOS model. The use of hardware interface API on RTOS model (described in Sec. 4.2) translates to the direct use of the device driver for the FPGA, resulting in about two order of magnitude

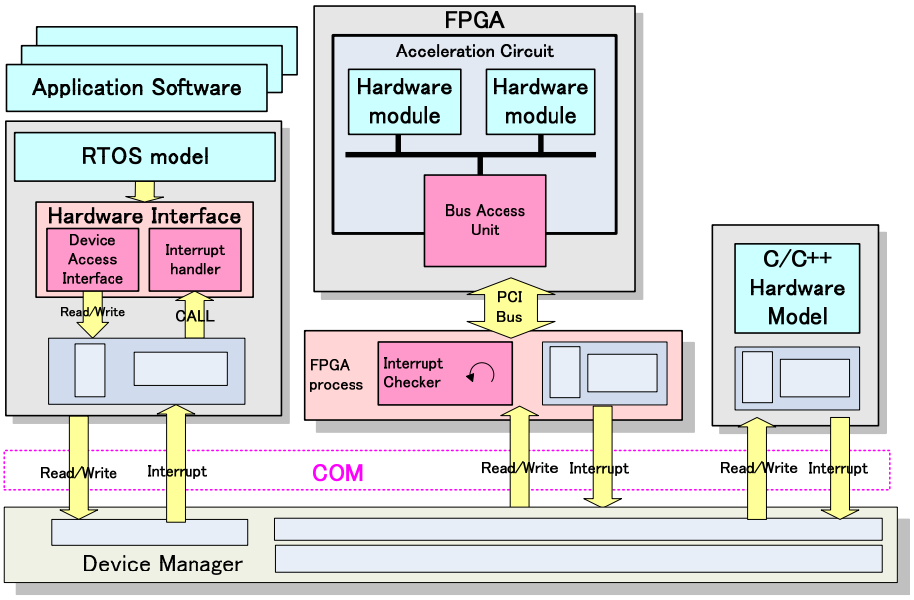


Figure 6.1: Covalidation using an FPGA by RPC-based communications

faster communication than using COM. This type of FPGA connection is useful especially for exhaustive validation with a large amount of test patterns. The direct connection is, however, less flexible than the COM-based connection. If connection is COM-based, the same device driver can be used independent of whether the hardware is executed on an FPGA or an HDL simulator, or even the hardware is a C++ model. Thus, hardware models can be easily replaced in a plug-and-play manner. If the FPGA is connected directly with the RTOS model, however, the device driver needs to be replaced as well. However, it should be stressed that, as described earlier, software programmers do not have to care about the type of FPGA connection at the application level.

For synchronization between hardware and software, our covalidation environment supports interrupts from hardware to software. In target systems, interrupts are performed immediately by interrupt signals at any time except when CPU is locked or the interrupts are masked (hence ignored).

In our covalidation environment, interrupts are handled differently between the HDL

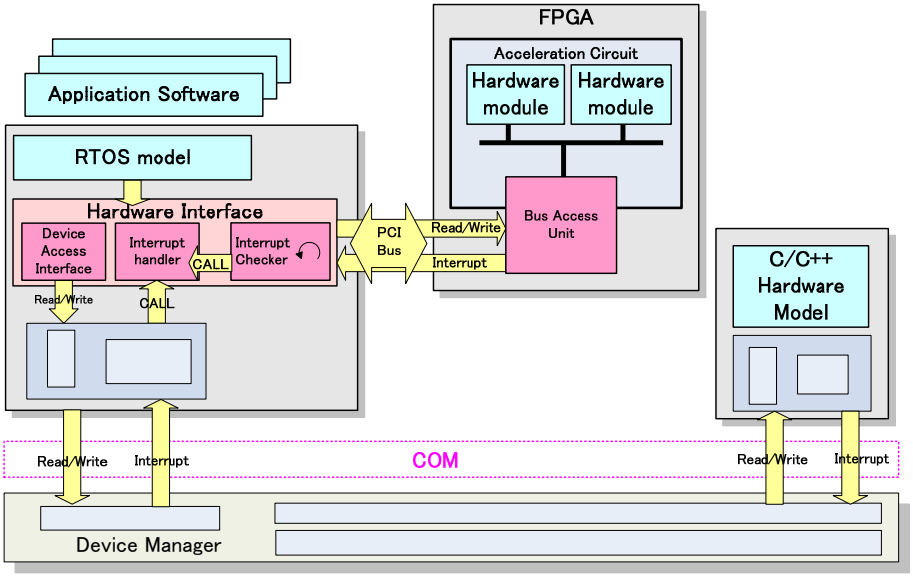


Figure 6.2: Covalidation using an FPGA by direct communications

simulator and FPGA. When the hardware modules are simulated on an HDL simulator, hardware interrupt signals are checked every clock rise in the HDL simulator, and an RPC of DM is called, so that covalidation environment can handle hardware interrupts immediately. When the hardware modules are executed on the FPGA, however, the FPGA process or the RTOS model cannot check interrupt signals from the FPGA at every clock timing of the hardware, because it cannot synchronize with the FPGA at clock timing level. Thus, the FPGA process or the RTOS model checks interrupts from hardware modules at a certain time interval (this function is denoted as *Interrupt Checker* in Figs. 6.1 and 6.2). Although this implementation causes a delay of a few milliseconds, the functionality of interrupts can be correctly performed.

6.3 EVALUATION OF COVARIDATION ENVIRONMENT

This section evaluates the effectiveness of our covalidation environment through a design of an MPEG4 decoder system. Experimental environments for this covalidation are shown

in Table 6.1. Note that we used an FPGA introduced in [27], which is connected to and accessed from the host computer through PCI Bus (denoted in Fig. 6.1).

Fig. 6.3 shows the design of the MPEG4 decoder system and allocation of tasks for the simulators. The MPEG4 decoder converts input data in the MPEG4 format into the YUV format, and writes the output data to a buffer of a video graphics array (VGA) device. The decoder system consists of a processor which executes application software with an RTOS, an acceleration circuit, and the VGA device.

The MPEG4 decoder application has four tasks; VLD, dequantization, IDCT and the others (denoted in Fig. 6.3). In the figure, the *Others* task covers any other tasks needed for MPEG4 decoder, e.g., decoder control, motion compensation and managing input and output data. The tasks executed on a processor are described in the C language. These tasks are compiled and linked together with the RTOS model to generate a binary code which is executable on the host computer. In order to shorten execution time of the MPEG4 decoder, some tasks are implemented as an acceleration hardware circuit. We used an MPEG4 file as input data which has 49 frames. Each frame size is 192×192 pixels, and the total file size is approximately 120KB. For the input data, each task of VLD, dequantization and IDCT is executed 7,766 times.

Table 6.2 shows elapsed times for covalidation for two system architectures. The second column of Table 6.2 shows covalidation time of the system where the IDCT task is implemented as an acceleration circuit. The third column of Table 6.2 shows that of the system where both dequantization and IDCT tasks are implemented as an acceleration cir-

Table 6.1: Experimental environments

Host CPU	Intel Core 2 Duo on 2.66GHz
Host main memory	2GB
Host OS	MS-Windows XP Professional
HDL simulator	ModelSim SE 6.1c
FPGA [27]	Spartan3 on 15MHz

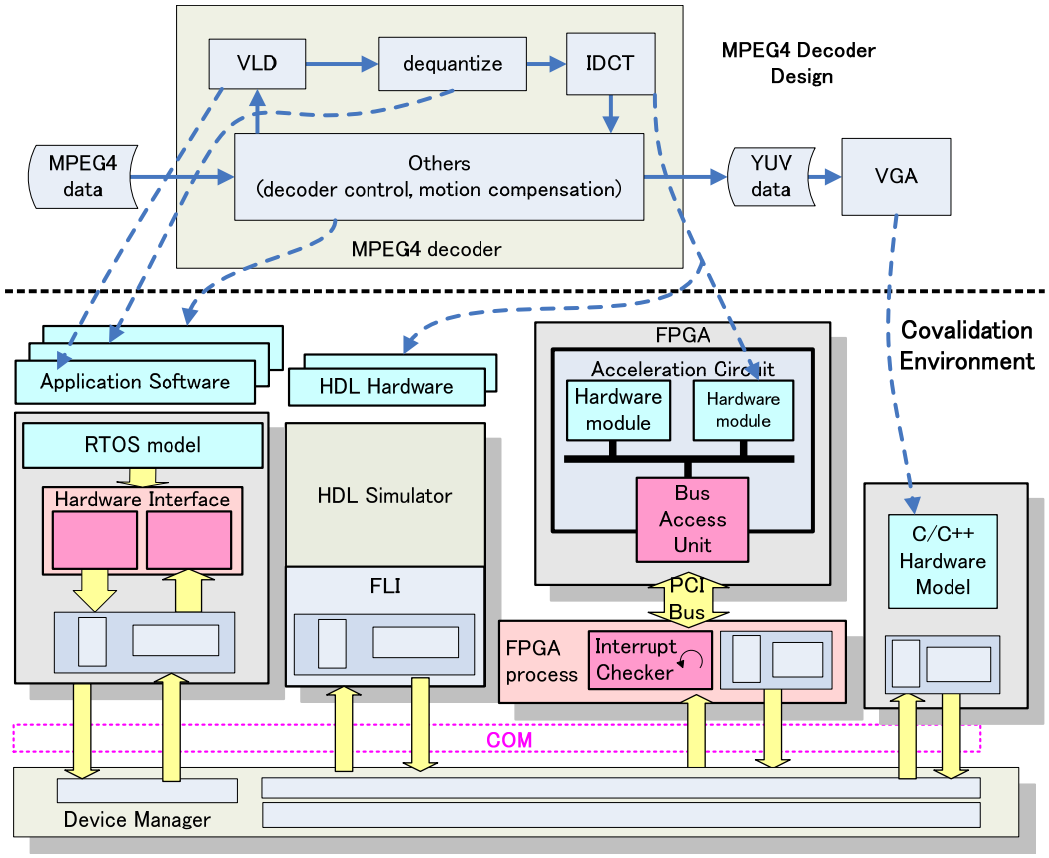


Figure 6.3: The MPEG4 decoder system and covalidation environment.

cuit. The acceleration circuit is written in HDL at the register-transfer level, and simulated on an HDL simulator or an FPGA. The VGA simulator is written in C++ and simulated in native execution.

A comparison of the second and fourth rows of Table 6.2 describes that our covalidation environment can perform software/hardware covalidation two orders of magnitude faster than the previous work [48]. Moreover, the second row shows that covalidation time is significantly affected by hardware complexity with the HDL simulator. On the other hand, hardware complexity has less effect on covalidation time when the hardware is emulated on the FPGA. These results shows effectiveness of our covalidation environment with RTOS model and FPGA.

Table 6.2: MPEG4 covalidation time

Hardwares	IDCT	dequant+IDCT
with HDL simulator[48]	5.36×10^2 sec	7.53×10^2 sec
with FPGA (COM)	1.48×10^2 sec	1.49×10^2 sec
with FPGA (Direct)	9.97 sec	6.25 sec

6.4 CONCLUSIONS

This chapter presented a software/hardware covalidation environment for embedded systems, which supports an RTOS model and an FPGA. The heart of our covalidation environment is the use of an RTOS model and an FPGA. The use of an RTOS model enables accurate system specification, efficient validation, and smooth implementation. In addition, the use of an FPGA enables performing covalidation in a short time. We also showed a case study in order to demonstrate the effectiveness of our covalidation environment.

CHAPTER 7

SYSTEM-LEVEL PROFILERS

7.1 INTRODUCTION

In order to design embedded systems of high quality in a short time, fast and accurate profiling and evaluation are musts for design space exploration.

Performance evaluation of mapping decisions requires timed descriptions. Recent system-level design tools provide automatic synthesis capabilities of timed descriptions from untimed descriptions and mapping [11][55][12]. These tools convert processes which are mapped on software into compilable program modules and processes which are mapped on hardware into synthesizable RTL circuits. Channels are converted into appropriate communication modules. The synthesized timed descriptions can be evaluated by simulation or FPGA-based prototyping.

A number of researches on simulation-based evaluation were conducted in the past. Aiming at functional verification of hardware-software systems, fast but inaccurate simulation techniques were proposed[48][58]. In contrast, cycle-accurate hardware simulation tools[40][41] were widely accepted for accurate but slow evaluation in an industrial domain. Fummi et al. proposed a cosimulation framework for both verification and

evaluation[59]. Their framework provides synchronization mechanisms, which vary in speed and accuracy, on communications between hardware and software. Designers, therefore, can use an appropriate mechanism depending on their needs (verification or evaluation). However, since speed and accuracy are incompatible with each other on simulations on a host PC, simulations are often inappropriate for design space exploration, especially for recent complex systems.

Another approach to performance evaluation is FPGA-based prototyping. FPGA-based prototypes achieve both high accuracy and speed, and are appropriate for iteration of mapping and evaluation. One disadvantage of FPGA-based prototypes is that internal states of the system are unobservable without additional modification for system descriptions. Since recent systems have complex dependencies and concurrency among processes, profiling capabilities are essential to find out bottlenecks and to help designers decide mapping alternatives. However, manual modification for profiling is time-consuming and error-prone. In order to prune design candidates efficiently and find the best choice quickly, automatic instrumentation for profiling is necessary.

We have developed two profilers, a process profiler and a memory profiler, for FPGA-based performance analysis of design candidates. The process profiler records a trace of process activations, while the memory profiler records a trace of memory channel accesses. In our framework, systems are described at a high level and FPGA-based system prototypes are automatically synthesized by SystemBuilder-MP. According to mapping of processes to PEs, the profilers are automatically configured and instrumented into the FPGA-based system prototypes by SystemBuilder-MP. Designers therefore need to manually modify neither the system description nor the profilers upon each change of process mapping. The profilers allow fast and accurate performance evaluation of the systems which have complex dependency and concurrency with the profilers using an FPGA. In summary, major contributions of our profilers on design space exploration are

- automatic instrumentation of the profilers with support of a system-level design tool,
- fast and accurate FPGA-based evaluation and profiling,
- and profiling capabilities for concurrent MPSoCs.

The rest of this chapter is organized as follows. First, Section 7.2 describes two proposed profilers. Next, Section 7.3 shows the effectiveness of the profilers through two case studies. Finally Section 7.4 concludes this chapter with a summary.

7.2 SYSTEM-LEVEL PROFILERS

We propose two profilers, a process profiler and a memory profiler. The profilers are automatically configured and instrumented into FPGA-based prototypes by SystemBuilder-MP, and record traces of processes and memory accesses at runtime of the prototypes.

The process profiler has been developed to help designers analyze behaviors of processes taking concurrency and dependencies among processes into account. Since processes of recent embedded systems may have complex dependencies with each other, the mapping decision which maximizes the parallelism of processors and dedicated hardware modules is not obvious and needs to be explored. In order to find the optimal mapping of processes, evaluation of mapping decisions is not sufficient with only execution time of individual processes. It is important to record activation/wait timings of processes and analyze the parallel behavior of processes with the timings.

Processes have not only explicit dependencies among them represented by blocking channels but also implicit relationships which appear in accessing shared resources. The memory profiler has been developed to help designers analyze the effects of such conflicts at memory channels. Recent embedded systems which consist of concurrent processes often share memories for communication and resource reduction. Since simultaneous mem-

ory accesses for a single memory module cause conflicts and need to be handled sequentially by bus arbiters or memory interfaces with additional cycles, they may cause performance degradation. Therefore the memory profiler records traces of memory channels to analyze conflicts on shared memories.

It should be noted that the profilers do not restrict the capability of SystemBuilder-MP. In other words, the profilers can be inserted into any system designed by SystemBuilder-MP. In turn, since the current profilers are tightly coupled with SystemBuilder-MP, the profilers can hardly be applied to system architectures which are not supported by SystemBuilder-MP. Another restriction at present is that at most 16 processes can be profiled by the process profiler and at most 16 memory channels can be profiled by the memory profiler, but this restriction will be relaxed in near future.

In this section, we describe advantages, profiling flow and detail of the profilers.

7.2.1 ADVANTAGES

In order to explore design candidates efficiently, we developed the profilers which have following advantages:

1. automatic instrumentation of the profilers at the system synthesis phase,

2. common timeline between software and hardware,

3. low profiling overhead on performance,

4. visualization of traces for intuitive analysis.

Automatic instrumentation of the profilers is necessary for realizing smooth iteration of mapping and evaluation. Otherwise, designers have to manually modify system descriptions and the profilers upon each change of process mapping.

Common timeline between software and hardware is required since processes are mapped onto either software or hardware. In order to help designers find bottleneck processes out from both software and hardware, the profilers must record traces of them in a common timeline.

Performance overhead of the profilers must be small enough not to have influence on behaviors of processes. We especially gave priority to minimize performance degradation on development of the profilers. This is why we decided to implement major parts of the profilers in hardware. We paid less attention for area overhead than performance overhead since the profilers are removed from the final implementation of the system in our design flow.

The objective of visualization of traces is to help designers find bottlenecks out from complex process behavior.

7.2.2 PROFILING FLOW

Fig. 7.1 shows the profiling flow of our profilers with SystemBuilder-MP. The profiling flow starts from "system-level description" (denoted in Fig. 7.1), and SystemBuilder-MP automatically generates an FPGA-based prototype according to a mapping decision of a designer. The prototype consists of "FPGA configuration" of hardware modules and "executable binary" of software, and the process profiler and the memory profiler are configured and instrumented in software and hardware automatically. Process trace and memory trace are recorded by executing the FPGA-based prototype. Then, "trace output" is transferred from the FPGA to the host PC. Finally, the trace output is transformed for analysis and visualization by "trace analyzer & visualizer".

After this flow, designers can analyze the prototype using the traces, and can make feedback for the inputs of SystemBuilder-MP to find better system implementations.

Since capacities of memories are limited, the profilers cannot record traces of an entire

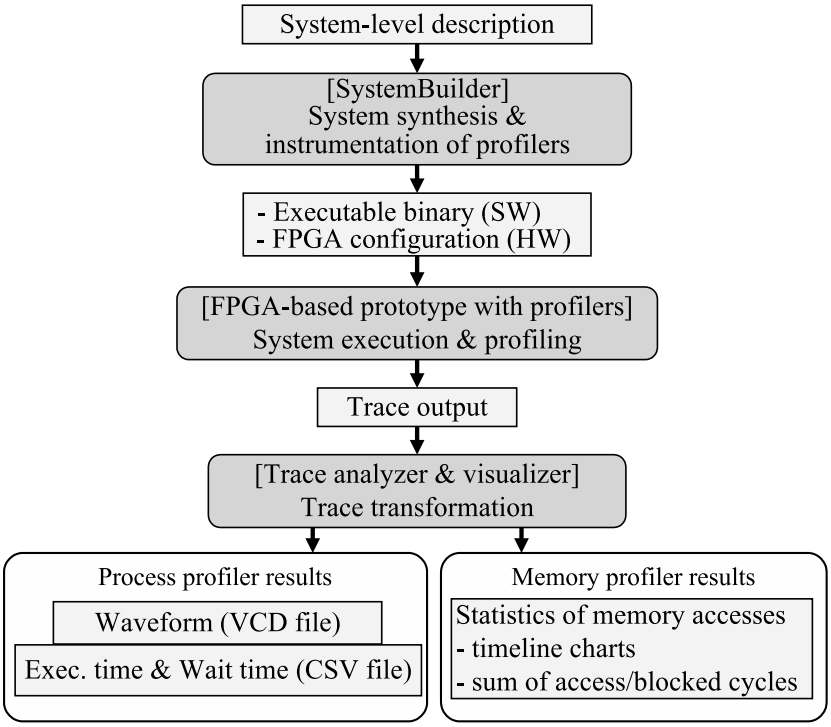


Figure 7.1: Tool flow of the profilers supported by SystemBuilder-MP.

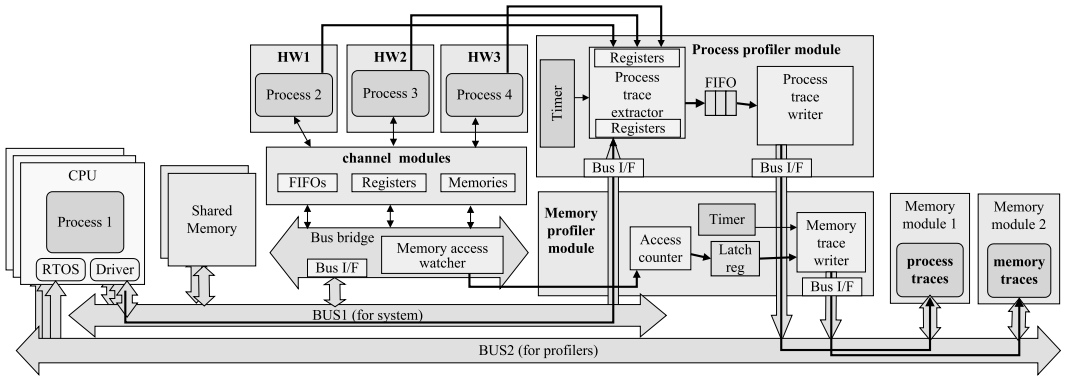


Figure 7.2: Overall structure of the process profiler and the memory profiler.

system execution. SystemBuilder-MP provides APIs to specify the timings where the profilers start and end. Designers write the API calls in any point of the process descriptions and get traces during the period they are interested in.

7.2.3 THE PROCESS PROFILER

The process profiler records a trace of activation/wait timings of processes through the execution period specified by a designer.

Fig. 7.2 illustrates the overall structure of our profilers. The process profiler consists of processes which are instrumented for profiling and "process profiler module" hardware (illustrated in Fig. 7.2). The process profiler module consists of "process trace extractor", "process trace writer", a timer module, a FIFO and a memory module. At runtime of the system, processes send signals to registers of the trace extractor. The process trace extractor collects the values of the registers, and sends them with a time-stamp obtained from the timer module to the process trace writer through the FIFO. The process trace writer writes data to the memory module whenever it receives data from the FIFO. Since we made a dedicated memory module for the process profiler and a dedicated access interface for the memory module, the memory accesses of the process trace writer do not conflict with other communications among processes, and have no effect on performance of the system.

All accesses for blocking channels, which are used to activate processes, are automatically transformed to send signals to the process trace extractor by SystemBuilder-MP. Fig. 7.3 shows an example of a transformed description which accesses a blocking channel (denoted as XXX_BC_READ). The transformed description consists of an original functionality which accesses the blocking channel (denoted as function calls of `yx_meschan_read()` and `syscall()` in Fig. 7.3) and signaling functionalities (denoted as two function calls of `profiler_set_state()`). By the calls of `profiler_set_state()`, the process writes "0" to a register of the process trace extractor at the beginning of the access, and writes "1" at completion of the access. The description in C is converted for a target processor and an FPGA by compilers and behavioral synthesis tools, respectively.

Note that computational results of the processes do not change between before and

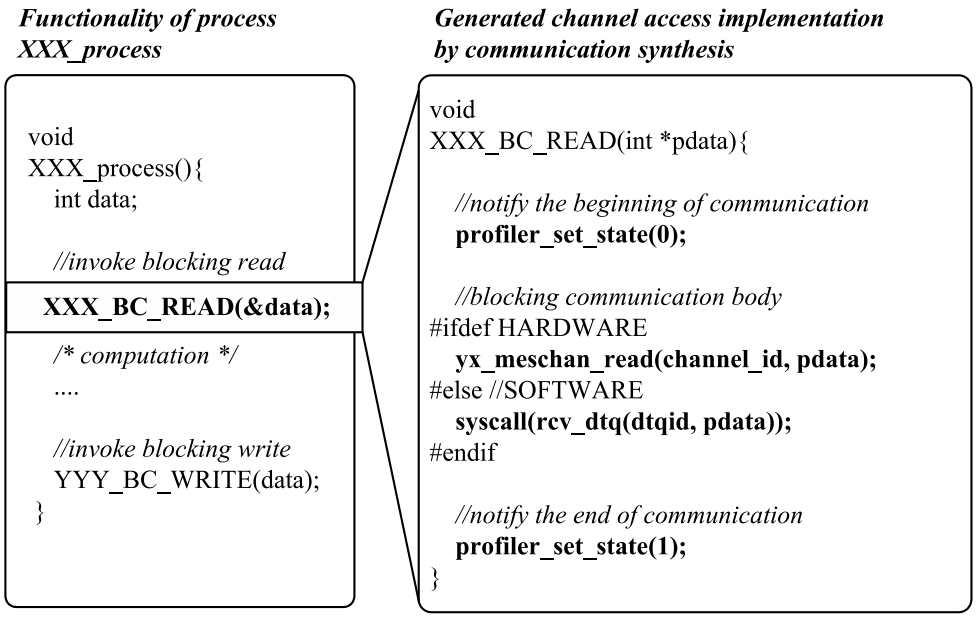


Figure 7.3: Instrumentation example of a process for profiling.

after instrumentation of the process profiler. This is because the instrumentation of the process profiler only adds the signaling functionality to processes. However, instrumentation of the signaling functionalities can result in degradation of performance and accuracy of FPGA-prototypes. In other words, execution time (i.e., execution cycles) of processes may increase by insertion of a process profiler, and therefore, the execution cycles of the processes with the profiler are not as completely same as the ones without the profiler.

Degradation of performance is inevitable for processes implemented in software (software processes, hereafter). Software processes should notify their states to the process profiler module with additional instructions since internal states of processors are not observable from other modules. Fig. 7.4 shows the program code of the signaling function described in C for software processes, which appears as "`profiler_set_state()`" in Fig. 7.3. In the function, a software process, which is implemented as a task of an RTOS, first obtains its task ID using an RTOS's API. Next, the process obtains an address of a register in the process trace extractor by using the task ID. Finally, the process writes its

```
 1: void profiler_set_state(unsigned char state){
 2:     int tskid;
 3:     unsigned char *reg4tskid;
 4:
 5:     //RTOS-API to get Task (Process) ID
 6:     get_tid(&tskid);
 7:
 8:     //get register address for this process
 9:     reg4tskid = TaskID2StateRegMap[tskid];
10:
11:     //signal state of the process
12:     //to the process trace extractor
13:     *(volatile unsigned char *)reg4tskid = state;
14: }
```

Figure 7.4: An example of the signaling function for software.

state (executing (1) or waiting (0)) to the register. Assembly code of this function for a Nios II processor consists of 50 instructions. Since the signaling function is called twice per blocking channel access, software processes need additional 100 instructions for each blocking channel access.

As for the processes implemented in hardware (hardware processes, hereafter), "`profiler_set_state()`" in Fig. 7.3 is done in a single clock cycle. This is because the implementation of the `profiler_set_state()` is realized by writing a state of the process to a register in a single clock cycle. Therefore hardware processes basically need additional two clock cycles for every blocking channel access. However, the signaling functionality and the original behavior of the process may be executed in parallel (depending on behavioral synthesis results). Consequently the overhead of the signaling functionality of hardware processes will be two or less clock cycles per blocking channel access.

After the execution, the trace data are read from the memory module and are output to a debug console on a host PC. The process profiler also provides the trace analyzer & visualizer for the traces obtained from an FPGA (illustrated in Fig. 7.1). The trace

analyzer & visualizer generates a VCD (Value Change Dump) file and a CSV (Comma Separated Value) file. The VCD file can be visualized as waveforms using tools such as *GTKWave* [60]. In the waveforms, high states mean executions of the processes and low states mean waiting times of them. Visualization of the system behavior can support designers to intuitively grasp complex parallelism of the processes. The CSV file contains formatted traces and can be fed by various tools for further analysis.

7.2.4 THE MEMORY PROFILER

The memory profiler records traces of shared memory accesses including access cycles and blocked cycles. Since the memory accesses are performed frequently and tend to cause exhaustion of memories for the traces, the memory profiler records the sum of the access/blocked cycles for every n cycles specified by designers in order to use limited memory capacity efficiently.

The recording part of the memory profiler is implemented in hardware (illustrated as "memory profiler module" in Fig. 7.2). We designed the memory profiler focusing the feature that all processes mapped on hardware are implemented to use interface for outside memories by SystemBuilder-MP (illustrated as "bus bridge" in Fig. 7.2). In order to record memory accesses, "memory access watcher" inside the bus bridge tells the occurrence of memory accesses to "access counter". The access counter records the sums of the access/blocked cycles of individual channels in a certain period, and sends the sums to "memory trace writer". The period is specified by designers using an API at the beginning of profiling. For each period, the memory trace writer sends the sums of access/blocked cycles to the dedicated memory module which stores the memory access traces. SystemBuilder-MP automatically configures the memory access watcher depending on mapping of memory channels onto shared memory modules, and instruments the hardware module with the memory profiler module.

After the profiling, the traces are read from the memory module and are transferred to a host PC. The trace analyzer & visualizer transforms the traces and generates various intuitive graphs which show statistical values of entire execution and periodic changes.

Here, we discuss the accuracy of the system between before and after instrumentation of the memory profiler. First, it should be noted that instrumentation of the memory profiler by SystemBuilder-MP does not change implementation of processes. Therefore computational results of processes are not changed by the instrumentation. Moreover, the memory profiler does not change cycle-level behavior of the FPGA-based prototype. The memory profiler capability consists of three modules, i.e., the memory profiler module, a memory module to store memory access traces and the bus bridge where the memory access watcher is embedded. None of the three modules changes cycle-level behavior of the prototype as follows.

The access counter in the memory profiler module does not block the behavior of the bus bridge since it only receives signals brought by the memory access watcher.

The memory trace writer in the memory profiler module and the dedicated memory module ("Memory module 2" in Fig. 7.2) are connected by a dedicated bus (denoted as "BUS2"), hence communications between them do not conflict with other communications among processes on "BUS1".

The memory access watcher actually only brings internal signals of the bus bridge to the access counter, so that it does not interfere with the cycle-level behavior of the bus bridge.

For these reasons, cycle-level behavior of FPGA-based prototypes is not changed by instrumentation of the memory profiler.

However, the number of clock cycles required to execute overall the system will increase because a software process needs to call APIs which start/stop the execution of the profiler. Nevertheless, cycle-level behavior of the system between start and stop of the

memory profiler, in which designers are interested, are accurate between before and after the instrumentation.

Currently the memory profiler can trace the memory accesses performed by the processes implemented on hardware. Designers therefore cannot analyze impacts of conflicts caused by the processes on software directly, while the designers can see the impacts from the blocked cycles recorded by the memory profiler which include conflicts caused by software. This is because we laid emphasis on automation of the profiling flow and compromised on limitations of a logic synthesis tool.

7.2.5 FURTHER DISCUSSION ON FPGA-BASED PROFILING

As discussed in 7.2.3 and 7.2.4, profiler instrumentation may change the performance (execution cycles) of processes. In addition, a general problem exists in our approach to FPGA-based prototyping and profiling, i.e., FPGA-based prototypes can hardly behave as same as final ASIC implementations. This problem comes from various reasons. For example, since the capacity of FPGA is generally much smaller than that of ASIC, multiple FPGAs need to be used and the interconnection delay between FPGAs arises; some IP components (such as memory) for FPGA behave differently from those for ASIC. This problem is not specific to the work presented in this dissertation, but arises in any FPGA-based prototyping, and hence, is out of scope of this dissertation. In other words, this work (i.e., SystemBuilder-MP with the profilers) does not provide any solution to this problem.

In fact, current SystemBuilder-MP further complicates this problem because it supports only Altera's Nios II processors and Avalon buses, which are rarely used in final ASIC implementation in industry. In order for SystemBuilder-MP to be used for prototyping of ASIC design in industry, SystemBuilder-MP should support processors and buses which are actually used in final ASIC implementation. To this end, we need not only FPGA-synthesizable IPs of the processors, buses and peripherals, but also synthesis tools and

software development toolkit, and integrate these IPs and tools into our SystemBuilder-MP design environment. This work is possible, but has not been realized yet.

7.3 CASE STUDIES

In this section, we demonstrate the effectiveness of multiple bus interface generation capability and our profilers, In addition, we evaluate performance/area overhead of the profilers.

The case studies were performed on the following environment. The systems were designed on a PC whose OS is Windows XP Professional with an Intel Core 2 Quad 2.66 GHz processor and 2 GB RAM. The target board has an Altera Stratix II FPGA with Nios II soft-core processors at 50 MHz of clock frequency. eXCite 3.2a [46] was used for behavioral synthesis. Logic synthesis and place-and-route were done by Quartus 8.1.

7.3.1 AES ENCRYPTION SYSTEM DESIGN

First, we demonstrate an application example of the process profiler on an AES encryption system. Fig. 7.5 shows the functional structure of the AES encryption system. The AES encryption system consists of six processes, *aes_main*, *encrypt*, *keyschedule*, *addroundkey*, *looppart* and *endpart*. The six processes are executable in a pipelined manner. Each process is connected to its successor process through a blocking channel. Blocking channels have enough buffers not to block sender processes. In order to evaluate performance of the system, we measured the execution time of the system by executing the encryption 100 times per evaluation.

Fig. 7.6 illustrates a design space exploration scenario in this case study. The purpose of this exploration is to find mapping which maximizes parallelism of the processes on a limited number of processors. We therefore explore mapping of the processes onto one or two processors, and present waveforms of processes in three different mapping specifica-

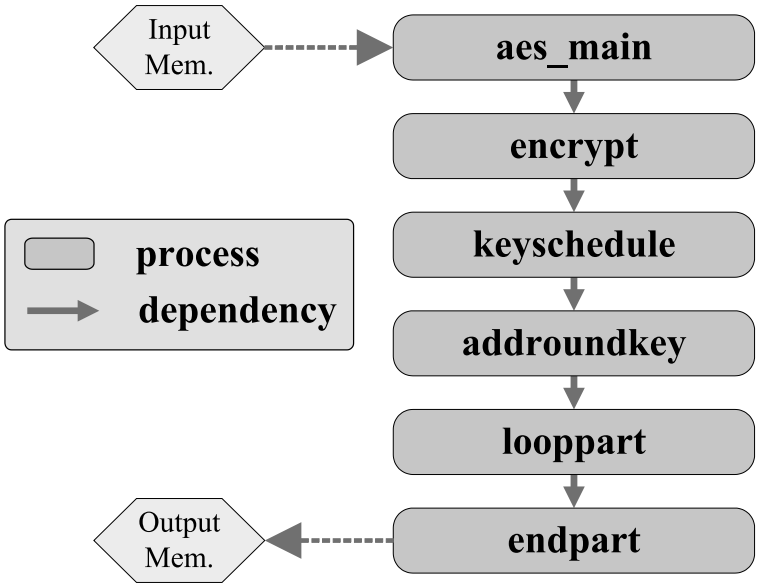


Figure 7.5: Functional structure of the AES encryption system.

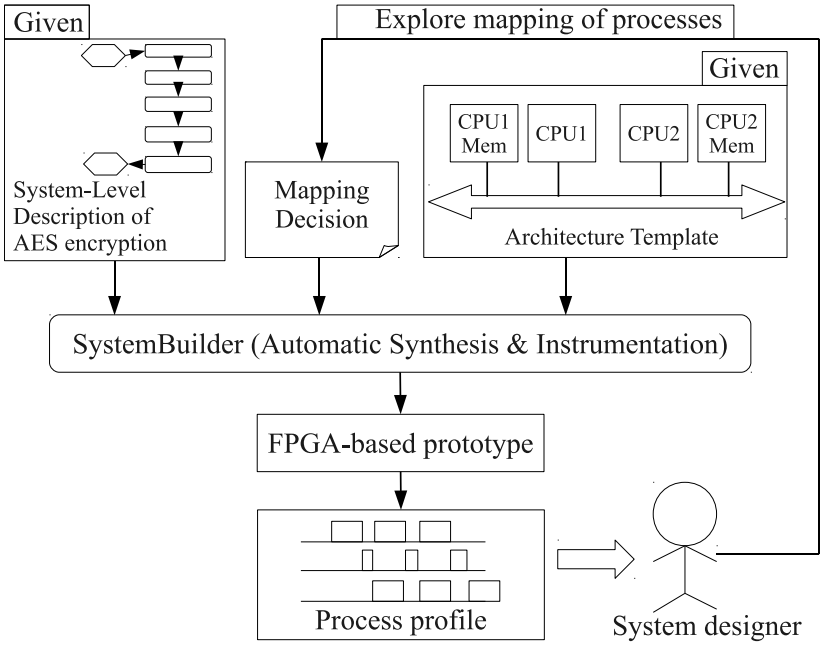
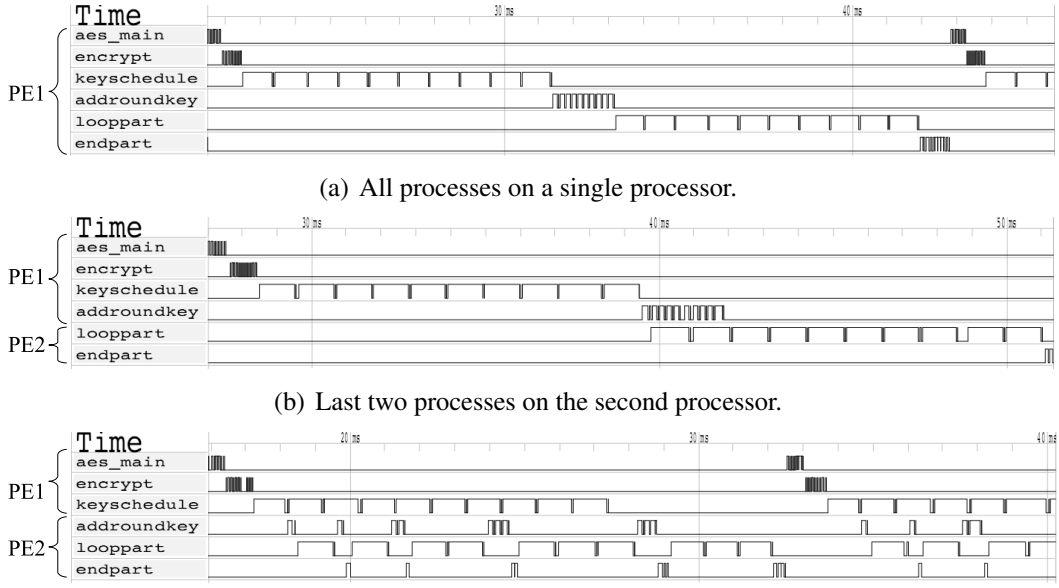


Figure 7.6: Design space exploration scenario of AES design.

tions. The memory profiler is not needed since we do not map any process onto hardware in this case study.



(a) All processes on a single processor.

(b) Last two processes on the second processor.

(c) "addroundkey" was moved to the second processor.

Figure 7.7: Process profiles of AES encryption system on three mapping decisions.

As a first step, we mapped all processes on a single processor to see the execution time of individual processes (in Fig. 7.7(a)). The total execution time of the system was 2,356 milliseconds. We could see that the keyschedule and the looppart processes consumed significantly longer execution time than others in Fig. 7.7(a). In order to improve performance, parallelizing the keyschedule process and the looppart process was expected to be obviously effective. We therefore changed mapping of the looppart and the endpart processes to the different processor.

Fig. 7.7(b) shows the waveforms of the processes after changing mapping of the looppart and the endpart processes. The total execution time of the system became 2,540 milliseconds. Contrary to our expectation, the total execution time was not improved and rather extended, even though the addroundkey process and the looppart processes were executed in parallel. This performance degradation might be caused by inter-processor communication. Moreover, the keyschedule process and the looppart process were not parallelized. The reason of this was obtained from Fig. 7.7(b). The figure shows that the addroundkey

process was not activated until the keyschedule completed its work. Since the addroundkey process was the activator of later processes, delaying activations of the addroundkey process led to the delay of the later processes, and prevented the processes from executing in parallel with the other processes. This was caused by the scheduling policy of an RTOS which the system employed. This is the good example showing that the processes whose execution time is shorter than those of others can become the bottleneck of the system and that the process profiler played an important role to find such bottleneck processes, since they cannot be found by comparing execution times of individual processes.

In order to execute the keyschedule and the looppart process in parallel, the addroundkey process must be activated immediately after an execution of the keyschedule process. It is achieved by mapping the addroundkey process onto the other processor. Fig. 7.7(c) is the waveforms of the processes after changing mapping of the addroundkey process. The total execution time of the system was 1,653 milliseconds. Fig. 7.7(c) shows clearly that the addroundkey process was activated immediately after an execution of the keyschedule process. As a result, the activation timings of the looppart process became earlier than before.

In this case study, the behavior of the processes are significantly affected by the dependencies among processes, the scheduling policy of an RTOS and the potential concurrency of processes, which cannot be considered by the analysis which relies only on the execution times of the individual processes. We could easily analyze such bottlenecks using the process traces. Thus we conclude that the effectiveness of the process profiler in exploring mapping of processes was proved.

7.3.2 MPEG4 DECODER SYSTEM DESIGN

We designed an MPEG-4 decoder system with SystemBuilder. Based on the MPEG-4 decoder developed in the past [61], we improved performance of the design using the process

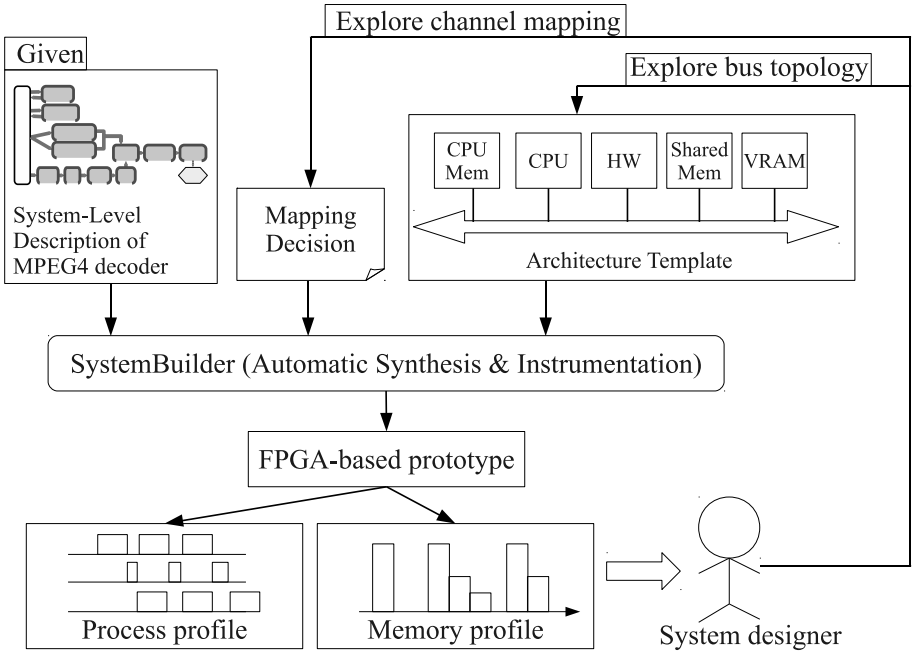


Figure 7.8: Design space exploration scenario of MPEG-4 decoder design.

profiler and the memory profiler (Fig. 7.8). This case study shows the effectiveness of both the process profiler and the memory profiler on a design refinement at system level.

Fig. 7.9 shows the structure of processes of the MPEG-4 decoder system. The MPEG-4 decoder consists of 12 processes, *mp4_main*, *header*, *get_mv*, *VLD*, *IQ*, *IDCT*, *catch*, *MI-1*, *MI-2*, *adder*, *yuv2rgb* and *display*. The mp4_main process handles inputs and the display process outputs decoded images to VRAM of a VGA device. The mp4_main process should be implemented in software, and the other processes can be implemented in software and hardware. So the term of "all hardware implementation" in this section denotes that all processes except for the mp4_main process are implemented in hardware. All processes can execute concurrently in a pipelined manner on all hardware implementation.

First, we explored several number of process mapping decisions and found that the all hardware implementation was the fastest implementation among them. However, its performance was yet 11.6 fps (frames per second) for 320 $\times$ 240 sized movies and needed

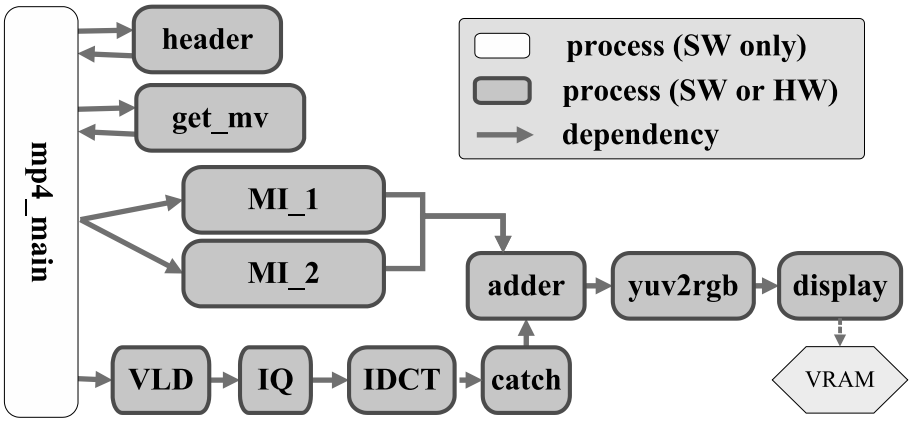
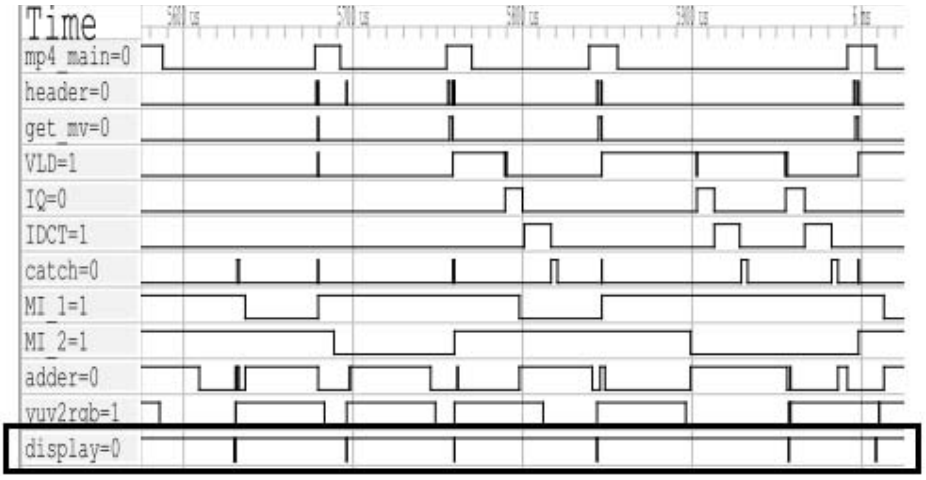


Figure 7.9: Functional structure of the MPEG-4 decoder.

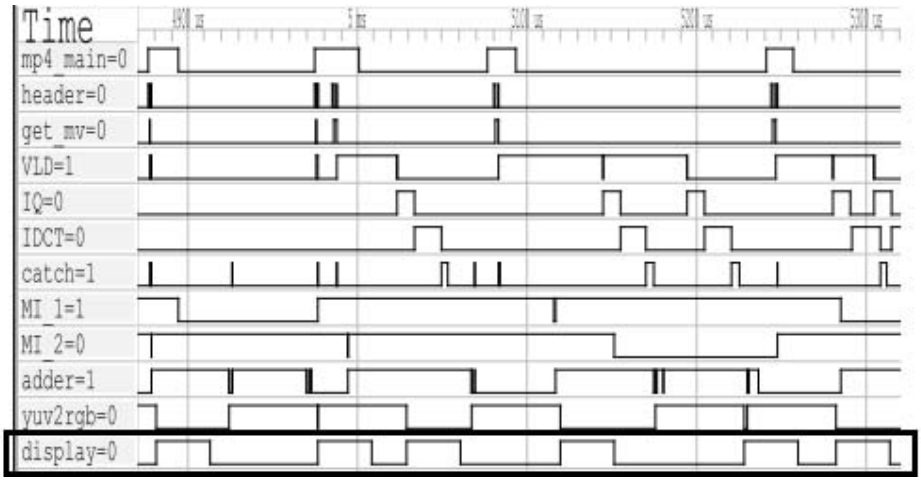
further improvements. We therefore used the process profiler in order to find the bottleneck processes. Fig. 7.10(a) shows the waveforms of the all hardware system. We could see that the yuv2rgb process (on the 2nd row from the bottom of the figure) could not be activated until the display process (on the bottom of the figure) finished its execution, and the display process was always active. In other words, the display process was a bottleneck. However, no solution was gained by reviewing the C program of the display process. We therefore used the memory profiler to obtain more information about the bottleneck.

Fig. 7.11 illustrates a graph obtained by the memory profiler. White bars on the back side of Fig. 7.11 illustrate the sums of blocked cycles (on y-axis) for individual channels which access off-chip memories (on x-axis) of the all hardware implementation. The right-most bar of Fig. 7.11 shows that the "VRAM_MEM_display" channel, which transfers decoded images to VRAM of the VGA device, was frequently blocked by other channels. This indicated that the execution of the display process was delayed by the conflicts and that the reduction of the conflicts may make the display process faster.

There were three points at which conflicts were possible to be caused: the bus, an interface of the VRAM for the VGA device and the bus interface of the hardware module (shown as "Bus bridge" in Fig. 7.2) which manages bus accesses from processes mapped



(a) MPEG-4 decoder with a single bus interface.



(b) MPEG-4 decoder with two bus interfaces.

Figure 7.10: The process profiler results of two MPEG-4 decoder implementations.

on hardware. Since the VRAM is accessed by the display process only, the memory accesses cannot conflict with others at the interface of the VRAM. We therefore concluded that the conflicts were caused at the bus and the bus interface. We solved the conflicts by making a special bus interface for the display process using SystemBuilder-MP's capability of multiple bus interface generation. Since the bus is implemented as crossbar switches in the FPGA, conflicts cannot occur at any point on accesses to the VRAM_MEM_display channel if the special bus interface is made. As a result, the special bus interface enabled

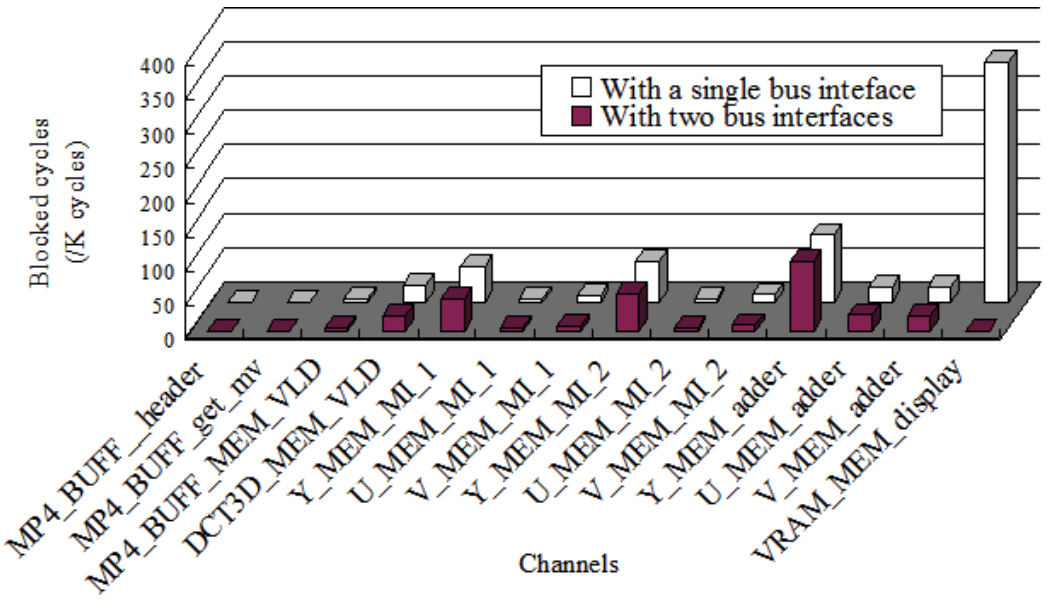


Figure 7.11: Sums of blocked cycles for each channels.

the display process to access the VRAM exclusively. We obtained a process profile shown in Fig. 7.10(b) and a memory profile shown as black bars in front side of Fig. 7.11. In comparison with the waveforms at the bottom of Fig. 7.10(a), the waveforms of Fig. 7.10(b) show that the execution time of the display process was reduced. The memory profile shown in Fig. 7.11 shows evidently that the blocked cycles of the VRAM_MEM_display channel were removed. In conclusion, we achieved to overcome the bottleneck on the display process with the profilers.

7.3.3 Overhead Evaluation of The Profilers

We evaluated overheads of our profilers on performance and cost. It should be noted that the overheads of the profilers do not influence the quality of the product. The overheads, which we evaluate here, only affect FPGA-based prototypes at a design space exploration phase.

Table 7.1 shows the performance overheads and Table 7.2 shows the area overheads

Table 7.1: Performance overheads of the profilers.

	w/o profilers	w/ process profiler	w/ memory profiler
AES (SW)	2,813 ms	3,067 ms (+9%)	-
MPEG-4 (SW)	28,240 ms	30,640 ms (+9%)	-
AES (HW)	561 ms	566 ms (+1%)	560 ms (+0%)
MPEG-4 (HW)	1,821 ms	1,825 ms (+1%)	1,840 ms (+1%)

Table 7.2: Hardware cost (ALUTs) of the profilers.

	w/o profilers	w/ process profiler	w/ memory profiler
AES (SW)	3,329	4,623	-
MPEG-4 (SW)	3,329	4,623	-
AES (HW)	15,144	16,434	17,237
MPEG-4 (HW)	34,680	36,002	37,932

on an FPGA. The first two rows of the tables show the overheads on the AES encryption system and the MPEG-4 decoder system in case all processes of the systems are mapped on a single processor as software. In the case, the process profiler brought 9% increase of the execution time of the systems. The overheads of the memory profiler are not shown in the case since the memory profiler has no means if there is no process on hardware. Last two rows show overheads of the all hardware systems. Because of parallelism of hardware, the profilers had fewer effects on performance and resulted in only 1% overhead for both systems. As for area consumption, additional ALUTs (adaptive look-up tables) are used for hardware part of the profilers. Table 7.2 shows that 1,200 or more ALUTs are necessary for the profilers.

We also evaluated synthesis time overhead brought by instrumentation of the process profiler and the memory profiler. The overhead on synthesis time may affect efficiency on design space exploration which is realized by iteration of changing inputs, synthesis and evaluation. Table 7.3 shows synthesis time of the systems shown in Table 7.1 and Table 7.2 with and without the two profilers. Synthesis time for the systems was measured at two synthesis phases: one is communication synthesis phase performed by SystemBuilder

Table 7.3: Required time for automatic instrumentation of the profilers.

	w/o profilers		w/ process profiler		w/ memory profiler	
	comm. syn.	HW syn.	comm. syn.	HW syn.	comm. syn.	HW syn.
AES (SW)	0.04 sec.	5 min.	0.04 sec.	6 min.	-	-
MPEG-4 (SW)	0.06 sec.	5 min.	0.06 sec.	6 min.	-	-
AES (HW)	0.29 sec.	33 min.	0.18 sec.	35 min.	0.29 sec.	37 min.
MPEG-4 (HW)	0.56 sec.	69 min.	0.57 sec.	74 min.	0.56 sec.	72 min.

("comm. syn." in Table 7.3), and the other is behavioral and logic synthesis phase performed by YXI eXCite and Altera Quartus ("HW syn." in Table 7.3).

As shown in Table 7.3, communication synthesis by SystemBuilder completed in a second even with the profilers, while behavioral and logic synthesis time was increased by several minutes which we believe is trivial compared with the overall design time.

7.4 CONCLUSIONS

In system-level design, system designers describe functionalities as processes and channels, and iterates mapping of processes onto processing elements and evaluation. We proposed two profilers for FPGA-based prototypes, a process profiler and a memory profiler. The process profiler records activation/wait timings of processes. The memory profiler records access/blocked cycles of shared memory accesses.

The profilers are automatically instrumented into the system by SystemBuilder-MP. Automatic instrumentation of the profilers enables smooth iteration of mapping and evaluation. Since evaluation is performed by FPGA-based prototyping, designers can evaluate design candidates fast and accurately.

We demonstrated the effectiveness of the profilers through two case studies on AES encryption system design and MPEG4 decoder system design. The AES encryption system design provided an example that the process profiler played an important role to find

bottlenecks caused by dependencies of the processes. The MPEG4 decoder system design presented performance refinements using the profilers considering conflicts at memory accesses.

Currently we are working for improving the memory profiler to record shared memory accesses from not only hardware but also software. Also, we want to extend the visualizing capability of the traces for more efficient analysis.

CHAPTER 8

A FAST PERFORMANCE ESTIMATION TOOL

8.1 INTRODUCTION

In order to design embedded systems of high quality in a short time, fast and accurate evaluation is musts for design space exploration.

In the multiprocessor system design, designers should explore and find a good design candidate which meets their requirements from a vast design space. In order not to miss the best mapping of a system, exhaustive exploration is ideal solution for design space exploration. However, performance evaluation, which is one of the most important evaluation for design space exploration, makes exhaustive exploration hard since it requires measurable amount of time for some kind of simulation or evaluation with FPGA-based prototypes. Therefore, fast performance evaluation technique is necessary.

Traditionally, various simulation methods and simulation tools are proposed. Although cycle-level simulation is promising technique which achieves high accuracy, it needs huge time to simulate a system and is not applicable for exploration by iteration of simulation.

There are fast simulation and performance estimation methods at a high level of abstraction for fast design space exploration. Trace-based simulations proposed in literature [35], [36], and so on use traces for estimating execution time of a system and abstract out details of execution in order to reduce estimation time. Although these abstract simulation methods can simulate/estimate performance of systems in short time, they need accurate traces. Moreover, these works do not mention how to obtain such traces.

In contrast, system-level design tools proposed recently uses FPGA-based performance evaluation. Such tools surveyed in [15] automatically synthesize implementation of FPGA-based prototypes from high-level description of systems. These FPGA-based methods are fast and accurate, and need not any traces for performance evaluation. However, exhaustive exploration of design candidates cannot be performed in a practical time since synthesis time of an FPGA-based prototype reaches one or several hours.

In order to incorporate accuracy of FPGA-based prototypes and speed of performance estimation, some of works provide integrated frameworks which combine automatic synthesis functionalities and fast performance estimation methods [38] [37]. These works provide overall design flow from high-level description of system to fast performance evaluation, including methods for obtaining performance models. Although their performance evaluation techniques are fast for design space exploration, they lack consideration of communication time among functions and therefore they are applicable to systems with limited architectures.

We propose a fast trace-based performance estimation tool, named SystemPerfEst. SystemPerfEst is combined with SystemBuilder-MP and forms an easy-to-use integrated framework for design space exploration of multiprocessor systems developed on an FPGA. SystemPerfEst uses a few traces obtained by our system-level profilers in order to develop performance models of systems and to obtain characteristics of the target FPGA. Then performance estimation of the system with other mappings can be done using the traces.

Characteristics of the target FPGA is used for estimating communication time among functionalities. Since communication time depends on FPGAs and RTOSs (or middlewares) running on it, they should be considered for accurate performance estimation. In other works which assume that there are databases which contain architecture characteristics such as [37], designers should develop such databases by hand and with detailed knowledge about the target FPGA and RTOSs. In contrast, designers can collect the characteristics of the FPGA with just a single synthesis and execution of the simple system description on the target FPGA in our framework. Therefore designers are not required detailed knowledge about the FPGA to develop such databases.

The estimation method of SystemPerfEst is fast, and the estimation results are accurate to the extent that they are comparable with evaluation results on FPGA-based prototypes. Note that the basic estimation mechanism used in SystemPerfEst is a traditional trace-based one like [38], [39] and [36]. However, our method can consider communication times spent by RTOSs and interruption handlers, and therefore our method is more accurate and applicable to wider architecture including RTOSs.

Moreover, since the input for SystemBuilder-MP and SystemPerfEst is common, designers can verify the exploration results on an FPGA, make feedback to the description of systems, and explore more mappings using SystemPerfEst again smoothly.

In addition to the performance estimation method, we propose stochastic models of bus arbitration delay in order to estimate performance of systems which involve conflicts on memory access. With the expected values obtained by the stochastic models, SystemPerfEst can consider the effects of arbitration delay on memory access keeping the estimation speed fast.

The contributions of this work are (1) integrated framework which combines a system-level design tool and a performance estimation tool, (2) performance estimation method for multiprocessor systems whose results are comparable with evaluation results of FPGA-

based prototypes, considering communication times spent by such as RTOSs and interruptions, and (3) stochastic models of bus arbitration delay.

Since our performance estimation method uses profiles of FPGA-based prototypes, target platforms of systems are limited to FPGAs and cannot be applied to design of ASICs. However, recent growth of FPGAs on their speed and capacity is raising them to the level of industrial products. Therefore our method can be used for such cases. Moreover, our method at least can be applied for design of ASICs for the purpose to prune out the obviously insufficient design candidates at early phase of design.

The rest of this chapter is organized as follows. First, Section 8.2 describes performance estimation method used in SystemPerfEst. Second, Section 8.3 proposes stochastic models of bus arbitration delay. Section 8.4 shows the effectiveness of the fast performance estimation method and stochastic models through a case study. Finally Section 8.5 concludes this chapter with a summary.

8.2 FAST PERFORMANCE ESTIMATION METHOD

This section describes detail of our performance estimation method used in SystemPerfEst.

8.2.1 APPROACH AND ASSUMPTIONS

In order to estimate the performance (total execution time) of systems fast and accurately, execution time of processes is the most important factor. Although the use of detailed simulation enables us to obtain them accurately, it will make estimation slow. Therefore, we abstract out computation part of process from process models and use execution time obtained from traces instead. Assuming that execution time of a process for an input on a PE does not change depending on mapping of other processes, execution time of the process will be the same as that in the trace of other mappings where the process is mapped

on the same PE. Although this approach can be applied for estimating performance of a system on various mappings, it should be noted that this approach can be applied only for the same inputs and the same amount of computation as the input traces are obtained. Also, we assume that all PEs are in a single clock domain and are driven in a single clock frequency.

Under the assumption above, if there are n kind of PEs, n traces are necessary and sufficient for performance estimation. For instance, in the design space exploration is done on the architecture where processes can be mapped onto one type of processors and dedicated hardware ($n = 2$), performance estimation needs only two profiles. One is a profile of mapping where all processes are mapped onto the processor, and another is a profile of mapping where all processes are mapped onto dedicated hardware. The usage of them are described in 8.2.3, 8.2.6 and 8.2.7. Obviously, our performance estimation method needs traces of processes obtained from FPGA-based prototypes generated by SystemBuilder or such system-level design tools. On the other hand, our method can be applied to systems targeting any FPGA with any processor if traces of processes can be obtained on them.

To make this assumption practically true, we assume that target processors have a sufficiently fast memory for their instructions and do not use any cache since the use of cache will make execution time of processors vary (described in 8.2.5).

In contrast, communication time among processes, especially time spent for blocking communications using FIFO channels, cannot be estimated simply using traces because of two factors. One is the implementation of channel APIs which depends on mapping of processes. Another is blocking time which changes depending on execution condition of a system.

Channel APIs, which are interfaces to channels, are implemented as RTOS API calls for software or as communication circuitry for hardware by SystemBuilder. Typically, a channel API call spends a constant amount of time depending on its implementation.

Therefore, time for channel API calls can be estimated using a database which stores time spent for channels. Such database can be developed before exploration (described in 8.2.9).

On the other hand, blocking time cannot be estimated from traces since it depends on execution condition of other processes and channels. Therefore, performance models in our estimation method manages the occupation of FIFO buffers and blocking time (described in 8.2.6).

8.2.2 ADVANTAGE OF OUR METHODOLOGY

Fig. 8.1 briefly shows an advantage on design space exploration of our framework over a traditional exploration methodology with FPGA-based evaluation method.

In FPGA-based evaluation method, designers iterate evaluation by synthesizing and executing FPGA-based prototypes for a number of mappings. In detail, an evaluation consists of six steps; system synthesis by SystemBuilder, software compile, behavioral synthesis, logic synthesis and P&R, FPGA configuration, and recording execution traces. Although part of these steps are independent, each step needs high computation power and they cannot be done in parallel practically on a host PC. Hence even a single evaluation of a mapping spends long time. It is therefore hard to perform design space exploration by iterating evaluation of a number of mappings in practical time.

In contrast, our framework only uses SystemBuilder at first. Once the profile of all software implementation and all hardware one is obtained, designers can explore a number of mappings by parallel execution of SystemPerfEst. Since SystemPerfEst is fast and can be executed in parallel on a host PC, design space exploration can be performed much faster than the FPGA-based evaluation method.

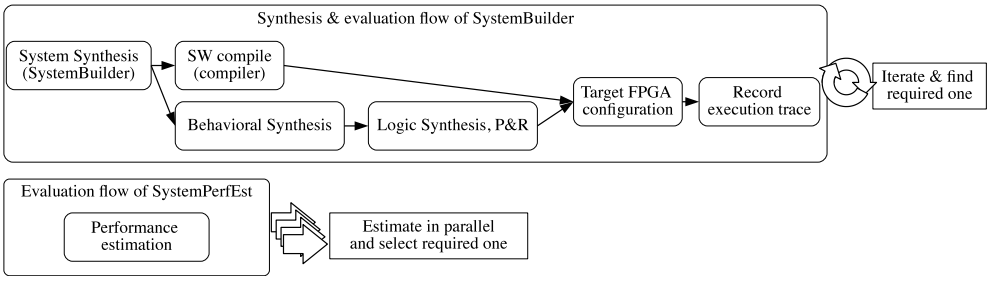


Figure 8.1: Comparison of evaluation time of SystemPerfEst with FPGA-based evaluation method.

8.2.3 CONCEPT OF OUR ESTIMATION METHOD

Before going details, we show a concept of our performance estimation method.

Fig. 8.2 illustrates the concept. Our method is a kind of trace-based simulation. We use two profiles obtained from FPGA-based prototypes as traces, one is a profile of all software implementation, and the other is that of all hardware implementation. From the two profiles, execution time (clock cycles) of processes on each computation step as software and hardware is obtained. According to mapping specification to be estimated, our estimation method selects profiles from all software implementation or all hardware implementation. After the selection of profiles, it calculates overall execution time (clock cycles) of the system considering parallelism of processes.

If the clock frequency of the system is determined, designers can calculate execution time (seconds) of the mapping by dividing the number of estimated clock cycles by the clock frequency. Although maximum clock frequency of a system generally may change depending on mappings, we leave the estimation method of them out of scope in this chapter.

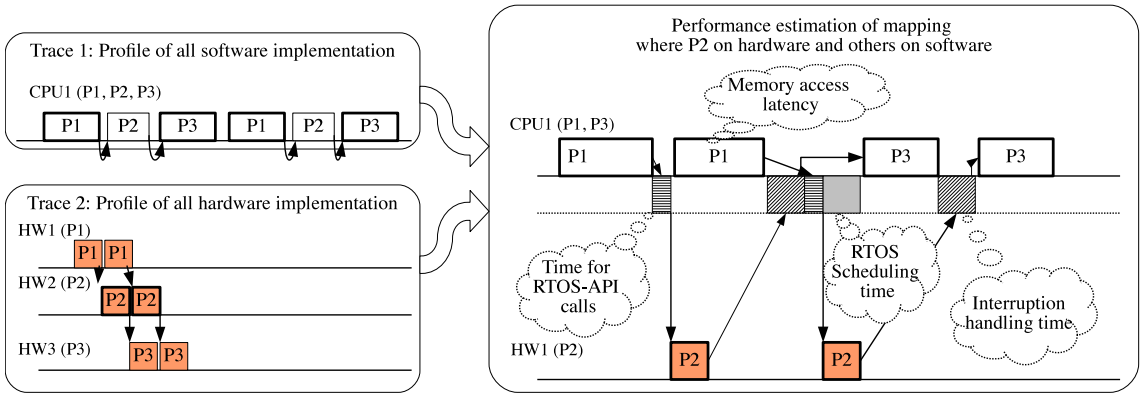


Figure 8.2: Concept of our performance estimation method.

8.2.4 OVERALL ESTIMATION FLOW

Next, we illustrate estimation flow focusing on inputs and outputs in Fig. 8.3. SystemPerfEst takes six inputs: (a) functional structure of a system, (b) channel record, (c)(d) profiles of two mappings (all software implementation and all hardware one) obtained by executing FPGA-based prototypes, (e) architecture characteristics, and (f) mapping specifications to be estimated.

Functional structure, profiles of a system and mapping specifications ((a), (c), (d) and (f)) are inputs and results of design flow of SystemBuilder. Therefore they are just fed to SystemPerfEst without any modifications.

Channel record (b) is a kind of event list which records invocation of channels by processes. Since process profiles do not contain records of channel invocations, SystemPerfEst needs them in order to know dependencies among processes and the number of memory accesses, which are determined in execution time of systems depending on its inputs. Channel record can be obtained from ISSs executing all software implementation of the system.

Architecture characteristics (e) are obtained from a profile of an FPGA-based prototype of a simple system description. The detail of architecture characteristics is described in 8.2.9.

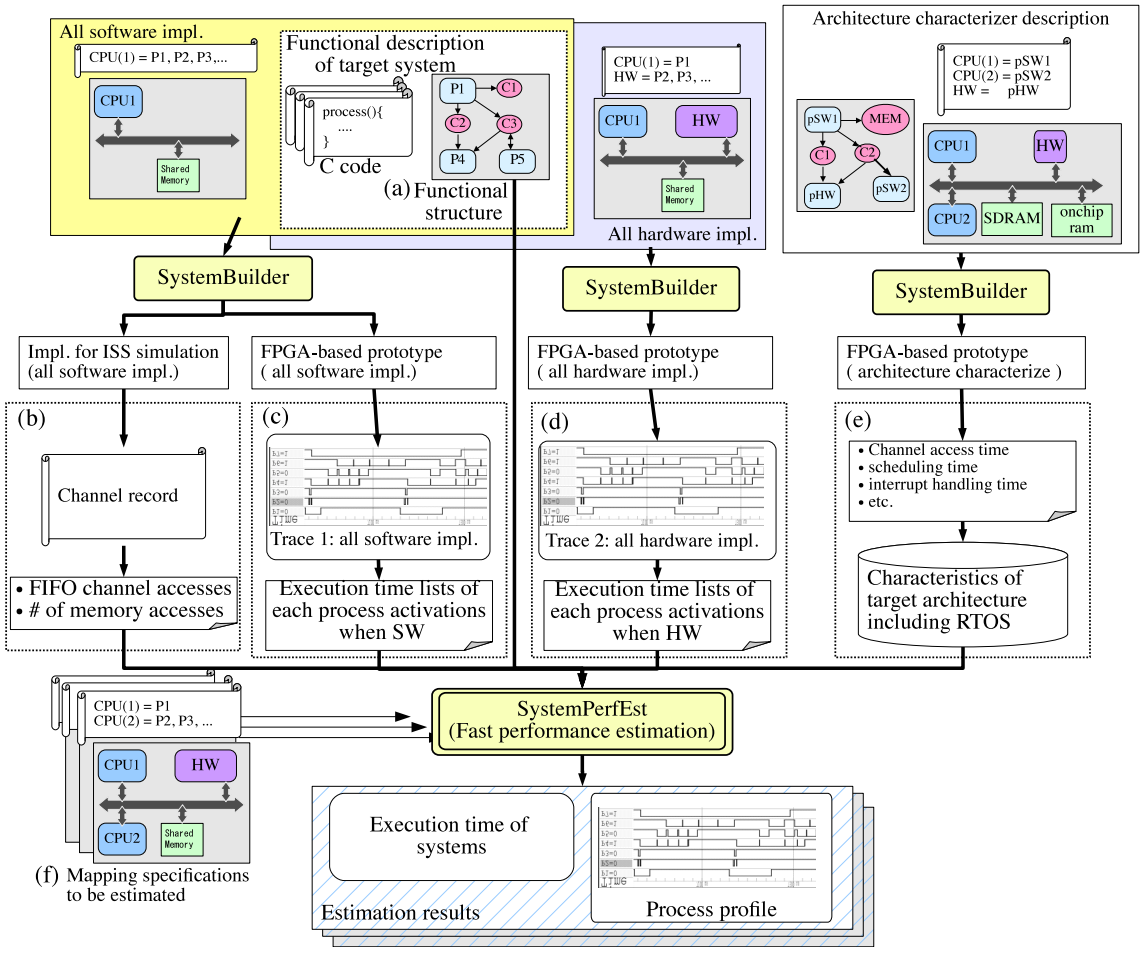


Figure 8.3: Inputs and outputs of SystemPerfEst.

As to the five inputs ((a), (b), (c), (d) and (e)), designers need to prepare only once at the first time of design space exploration of a system on a target FPGA.

With the six inputs, SystemPerfEst estimates performance of a design candidate described in mapping specifications (mappings). As the result of performance estimation, designers obtain execution time of systems and profiles which are in the same format as those of the process profile obtained from FPGA-based prototypes designed by SystemBuilder. After estimation of a number of mappings, designers can choose mappings which meet their requirements and/or analyze bottlenecks with the estimated profiles.

8.2.5 TARGET ARCHITECTURE

Fig. 8.4 shows a target architecture which SystemPerfEst can deal with. A single kind of processors is assumed. Under this assumption, system designers only need two traces (all software implementation and all hardware one).

The number of processors and dedicated hardware modules where processes are mapped can be changed and explored. Processors are assumed to have their own TCMs (tightly coupled memories) for instruction/data of software and have no cache. This is because of our assumption described in 8.2.1. Consideration of caches is one of our future works.

A single clock frequency among processors and hardware modules is assumed.

Designers can also explore memory modules used in their system such as on-chip SRAMs and off-chip SDRAMs which vary latencies of them.

Currently, consideration of buses such as bus protocols and bus topologies is not included. In our estimation method, we assume architectures which use a single virtual bus where neither burst transaction nor memory/bus conflict occurs. However, effects of bus latencies on performance are included in memory latencies obtained from architecture characterization results (described in 8.2.9).

8.2.6 PERFORMANCE MODELS

Fig. 8.5(a) shows performance models which SystemPerfEst defines. There are two groups of models, functional models and architectural models. Functional models represent processes, channels and RTOSs which are executed on PEs. Architectural models represent PEs which are used by functional models as shared resources. This separation of models is similar to works [38] and [39].

All functional models have their execution time list and two states, active state and wait state. In active state, they consume their execution time (that is, top of execution time

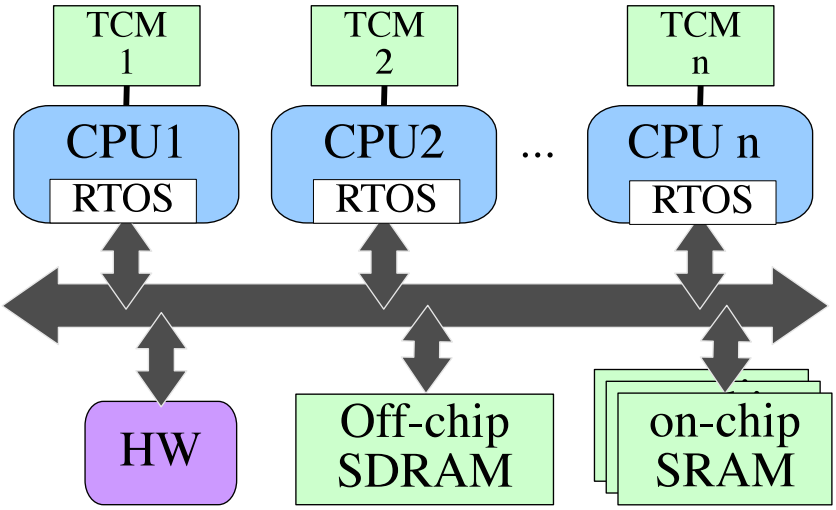
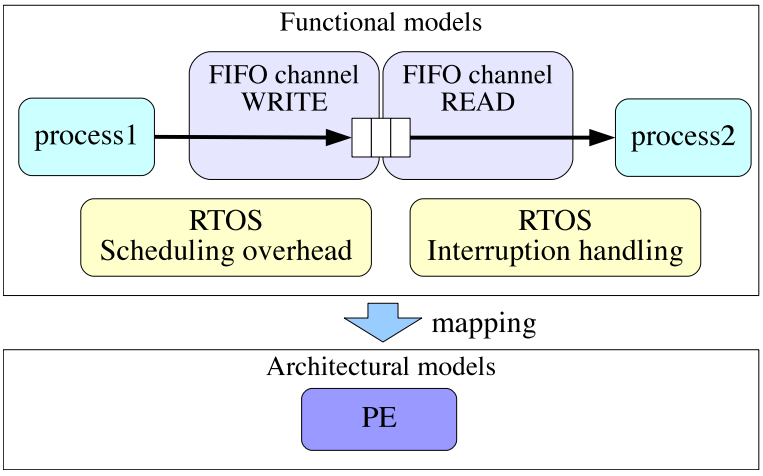


Figure 8.4: Target architecture which our performance estimation method can be applied.

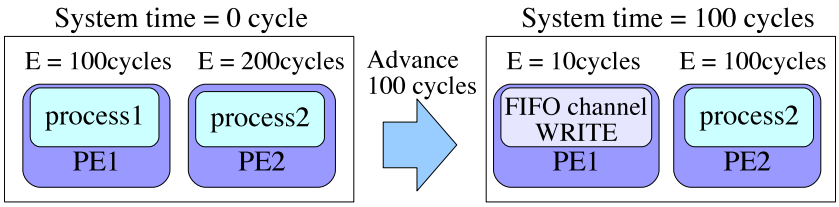
list) when they obtain the right to use PEs they mapped onto. When their execution time becomes zero, they delete the top of execution time list and change into wait state. In wait state, they do nothing but wait for events to change into active state.

A process model is a functional model which represent a process in functional description of a system. Execution time list of processes is an input trace of the process selected from profiles according to their mapping. A completion of an active state activate corresponding FIFO channel model. Since FIFO channels which should be activated vary depending on inputs for a system, process models also need FIFO channel call list which are obtained from channel record. In a wait state, process models wait completion of FIFO channel access.

A functional model of a FIFO channel (a FIFO channel model) manages API call time and buffer occupation. FIFO channel models consume their API call time when they are on an active state. The completion of an active state means completion of channel access and activates corresponding processes. Execution time of active state of FIFO channel models is determined using architecture characteristics. In a wait state, FIFO channel models wait



(a) Performance models defined in our performance estimation method.



(b) Simple example of performance estimation with performance models.

Figure 8.5: Performance models and estimation method using them.

invocation of them by process models. The number of buffers can be set and explored by designers. In multiprocessor systems, API call time for send and receive may occur on different processors on inter-processor communications. We therefore made FIFO channel apart to two parts, send part and receive part (shown as “FIFO channel WRITE” and “FIFO channel READ” in Fig. 8.5(a)).

There are two types of RTOS models, “Scheduling overhead” model and “Interruption handling” model. They manages scheduling overhead of RTOSs and interruption handling time used for FIFO channel between software and hardware, respectively. Execution times of them are determined by architecture characteristics. Both of them only time consumption on their PEs. Scheduling overhead model is activated when a process model on a PE changes. Interruption handling model is activated when a process on a processor is

activated by a process on hardware after blocking.

PE models are architectural models and manage processes which can consume their time. In other words, PEs schedules functional models. PE models acts like OSs and have their scheduling policies. Preemption is also supported.

For instance, a PE which represents a processor have a priority queue which stores executable processes and channels, and selects one along with scheduling policy. Currently, SystemPerfEst supports fixed priority-based scheduling which have been mostly used in RTOSs for embedded system. Exploration of scheduling policies is one of our future works.

In this sense, functional models are similar to tasks on RTOSs and therefore they have priorities. Priorities of process models can be set and explored by designers. By default, all process models have same priorities. As for FIFO channel models and RTOS models, they have highest priority in order to simulate scheduling of RTOSs.

8.2.7 PERFORMANCE ESTIMATION METHOD

On performance estimation, SystemPerfEst manages system execution time which starts from zero, and increments along with process execution time obtained from profiles. Functional models consume their execution time only when they could obtain the right to use their PEs. When an execution time of a functional model become zero, its state changes to wait state and activates related functional models. After the iteration of this, the estimation ends with the system time at the moment as the resulting estimated execution time, when all execution time lists of functional models are consumed.

Fig. 8.5(b) illustrates an example. In the figure, there are two PE models (“PE1” and “PE2”). First, the system time is zero. At that time, process models “process1” and “process2” have the right to use the PEs (shown in left side of the figure). Their execution time are 100 cycles and 200 cycles, respectively. Then SystemPerfEst increments system time 100 cycles (minimum of execution times of functional models), and processes consume

100 cycles of their execution time. Then process1 lose the right to use PE1 and become wait state. At the same time, next functional model "FIFO channel WRITE" obtains the right to use PE1 as a result of scheduling of PE1.

8.2.8 REFLECTION OF MEMORY ACCESS LATENCIES

In order to explore memory mapping, SystemPerfEst can reflect changes of memory access latency. In input profiles, traces of processes include memory access time on a certain memory mapping. Therefore SystemPerfEst changes execution time of process obtained from traces when designers try to estimate performance with different memory mapping.

Assuming that memory access latency to a memory module is constant, we can obtain execution time of process with a new memory mapping E_{new} from that with memory mapping in input profile E_{input} by $E_{new} = E_{input} + (L_{new} - L_{input}) \times N$. L_{new} and L_{input} represent latency of a new memory module and memory module used on input profile, respectively. N indicates the number of accesses to the memory region mapped on the memory module.

8.2.9 ARCHITECTURE CHARACTERIZATION

SystemPerfEst considers communication time among processors and memory latencies. Such communication time depends on characteristics of a target FPGA and an RTOS executed on it. In order to take such characteristics into account, our framework provides a description of a simple system for architecture characterization (hereafter, architecture characterizer description). The description consists of some processes and channels and their mapping. By only synthesizing and profiling the description on a target FPGA using SystemBuilder, designers can obtain the database of characteristics of the target FPGA easily (shown as (e) in Fig. 8.3).

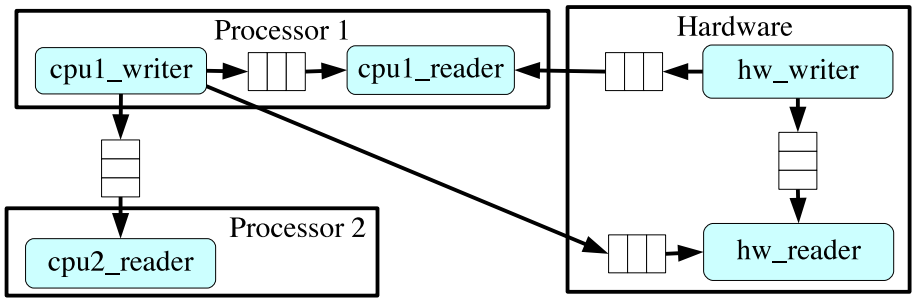


Figure 8.6: Functional structure and mapping of the architecture characterizer description.

A structure of functional description and their mapping is shown in Fig. 8.6. Mapping of processes and channels are fixed. Channels in the system are implemented differently by SystemBuilder with the mapping. For instance, a channel between processes "cpu1_writer" and "cpu1_reader" is implemented as an RTOS API for inner-processor communication, while a channel between processes "cpu1_writer" and "hw_reader" is implemented as communication between software and hardware.

These processes also access memories in a target FPGA, and collect latencies of them. The collected memory latencies include latencies of buses among PEs and memories.

With the architecture characterizer system, designers can make database which contains following time; inter-process communication time (inner-processor, inter-processor and processor to HW), memory access latencies (an off-chip SDRAM, an on-chip SRAM) and RTOS time (scheduling overhead, interruption handling).

One of good example of advantage of architecture characterization is estimation of systems with different RTOSs from systems as input. For example, SystemBuilder generates TOPPERS/JSP kernel[62] for single processor systems. On the other hand, as for multiprocessor architecture, SystemBuilder generates TOPPERS/FDMP kernel[62], which is a derivative of TOPPERS/JSP kernel for multiprocessor systems.

Since implementation details of communication APIs provided by the two RTOSs are different, communication time consumed by the APIs also differ. With architecture char-

acterization, SystemPerfEst accurately estimate communication time for both systems on single processor architecture and multiprocessor architecture.

8.3 STOCHASTIC MODELS OF BUS ARBITRATION DELAY

This section presents stochastic models of bus arbitration delay on simultaneous accesses to a shared memory from multiple processors. Since recent embedded systems have multiprocessors which run in parallel, our fast performance estimation tool should consider the effects of bus arbitration delay in short time.

We developed stochastic models for three bus arbitration policies, FCFS (first-come first served), FP (fixed-priority) and RR (round-robin). Our stochastic models can produce expected value database of bus arbitration delay. The database can be used for efficient consideration of bus arbitration delay in our fast performance estimation tool. Moreover, our stochastic models can be used to obtain probabilities of worst case delay.

We verified our models by comparing with Monte-Carlo simulation results.

8.3.1 ASSUMPTIONS ABOUT MEMORY ACCESSES

We made following assumptions about memory access situations for constructing the stochastic model. Memories are accessed by multiple processing elements (PEs) such as processors. The latencies of memory modules are constant and consume a unit of time. Also, the probabilities of all memory access of PEs are assumed to be equal. The stochastic models of bus arbitration delay consider memory access conflicts which occur in a time region which is defined as the reciprocal of the probability. In the region, each PE accesses the memory module once.

With above assumptions, PEs can be handled with no distinction. Thus we focus on the delay of a PE, which we call PE in focus (PIF) hereafter. PIF is assumed to issue its

memory access at the time 0.

Memory accesses are issued by PEs. If there is an access which is already granted for access, other accesses wait for end of the access. After that, arbiter of a bus or a memory module grants one of waiting accesses according to its arbitration policy. We call the wait time the bus arbitration delay.

We constructed stochastic models of the bus arbitration delay d of PIF. The models take the number of PEs except for PIF n and their access probability $f(t)$ as parameters. t represents time.

In order to simplify the model, we assumed that $f(t)$ forms uniform distribution (therefore $f(t) = f$).

8.3.2 AN EXAMPLE OF MEMORY ACCESSES UNDER THE ASSUMPTION

Here we show a calculation example of probabilistic distribution of bus arbitration delay. Fig. 8.7 illustrates an example with four PEs. The down arrows in the figure denote issue timings of memory accesses and the rectangles denote access times.

In the figure, the access from PIF occur at the time 0 and other processors access the memory module beforehand. In the memory accesses which occurred in $[-3, -2]$, "access 1" is granted first and "access 2" is granted secondarily after the completion of the access 1. The completion time of access 2 is in $[-1, 0]$ since the access latencies are an unit of time. In the access time of access 2, issue of access 3 occurs and it is granted at the completion of access 2. In this situation, access time of access 3 crosses over the time 0 where the memory access by PIF is issued. As a result, the memory access by PIF conflicts with access of access 3 and waits for its completion. The wait time is denoted as d in the figure. Our stochastic model of bus arbitration delay calculates probabilistic distribution of d.

Hereafter, we propose three stochastic models about three arbitration policies: FCFS, FP and RR. Stochastic models of FP and RR is constructed based on the FCFS model.

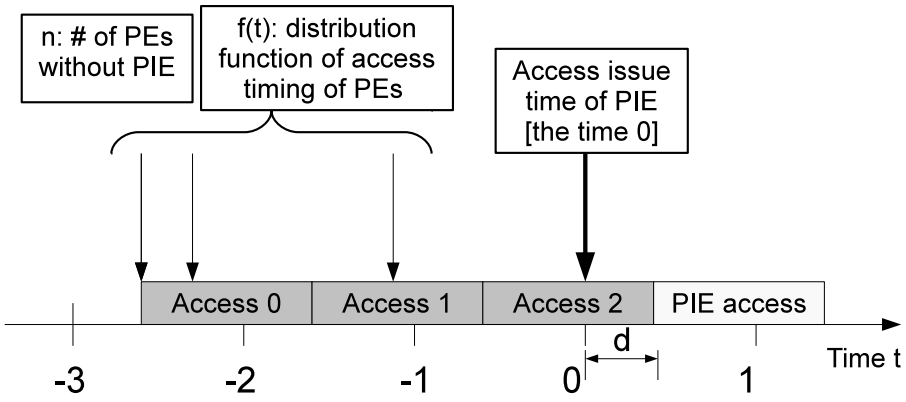


Figure 8.7: Assumed situation of memory conflicts.

The full definition of stochastic models are shown in Appendix B.

8.3.3 THE FCFS MODEL

In the FCFS arbitration, accesses are granted according to the order of their issued time. In this arbitration polity, accesses issued after the time 0 (the time when PIF issues its access) have no effect on the delay of the access by PIF. Therefore we consider only the time region before the time 0 and set the time region to consider as $[-1/f, 0]$.

We first show an example of calculation using Fig. 8.7. The access by PIF waits for the completion of access 3 for time d. There are two situations which correspond to such situation. (1) One is illustrated in the figure. There are more than two issues of memory accesses before the time -1 and only a single access of them crosses over the time 0. (2) In the other situation, there is only an issue at the time -1+d and no other access cross over the time -1+d. The probability of the situation (1) is considered as the situation where an virtual access issued at the time -1+d and the virtual access waits for a time period 1+d. The probability of the situation (2) is considered as multiplication of f(-1+d) by the probability of the situation where there are no access which cross over the time -1+d. The probability with no access cross over is calculated as complement of probabilities of the situations

where one, two,. . . , n accesses cross over the time -1+d.

With above way of thinking, we constructed the stochastic model of FCFS $W_{FCFS}(f, n, k, t, d)$. f, n, k, t and d denote the probability of memory access of PEs, the number of PEs except for PIF, the number of accesses which cross over the time t, the time when the memory access is issued by PIF, and the wait time of the access by PIF, respectively. $W_{FCFS}(f, n, k, 0, d)$ calculates the wait time of the access by PIF issued at time 0.

$I_{FCFS}(f, n, k, t)$ is a support function which provides the probability of the situation where k memory accesses cross over the time t. $I_{FCFS}(f, n, k, t)$ is used for construction of $W_{FCFS}(f, n, k, t, d)$ and models for FP and RR.

The definition of $W_{FCFS}(f, n, k, t, d)$ is as follows. In the definition, x represents values after the decimal point of d (thus $x = (d - (k - 1))$).

$W_{FCFS}(f, n, k, t, d) =$

$$
\begin{cases}
I_{FCFS}(f, n, 0, t), & k = 0 \\
n \times f(t-1+x) \times \left(\int_{t-1+x}^{t} f(\tau)d\tau\right)^{n-1}, & 0 < k = n \\
W_{FCFS}(f, n, k+1, t-1, k+x) & \\
\quad + \sum_{i=1}^{k-1} \left(\binom{n}{i} \times W_{FCFS}(f, n-i, k-i+1, t-1, k-i+x) \times \left(\int_{t-1}^{t} f(\tau)d\tau\right)^{i}\right) & \\
\quad + \binom{n}{k} \times W_{FCFS}(f, n-k, 1, t-1, x) \times & \\
\qquad \sum_{j=1}^{k} \left(\binom{k}{j} \times \left(\int_{t-1}^{t-1+x} f(\tau)d\tau\right)^{j} \times \left(\int_{t-1+x}^{t} f(\tau)d\tau\right)^{k-j}\right) & \\
\quad + \binom{n}{k} \times \left(\int_{0}^{x} W_{FCFS}(f, n-k, 1, t-1, \tau)d\tau + I_{FCFS}(f, n-k, 0, t-1)\right) & \\
\qquad \times k \times f(t-1+x) \times \left(\int_{t-1+x}^{t} f(\tau)d\tau\right)^{k-1}, & 0 < k < n
\end{cases}
$$

$$where\ x = d - (k-1)$$

8.3.4 THE FP MODEL

Next, we show the stochastic model of FP arbitration $W_{FP}(f, p, n, k, t, d)$. p means the priority of PIF on arbitration. $p \in [0, n]$ and p = 0 means the highest priority.

CONCEPT OF THE FP MODEL

We show the overview of model construction. The difference between FP and FCFS is that the access by PIF can be interfered by accesses which issued after the time 0. Therefore the FP model should consider accesses issued not only before the time 0 but also after the

time 0.

In order to consider accesses after the time 0, the FP model sets the time region to consider to $[-1/(2f), 1/(2f)]$ instead of $[-1/f, 0]$ in the FCFS model.

The access by PIF should wait until completion of successive accesses which cross over the time 0 and which with higher priorities issued after the time 0.

In followings, we explain the model of FP dividing to three cases.

CASE 1: MODEL OF THE LOWEST PRIORITY

When PIF is in the lowest priority, all other accesses issued after the time 0 may interfere the access of PIF.

Lets consider the situation where the access by PIF is interfered by k accesses. Assuming that $k1$ ($< k$) accesses issued after the time 0 interfere the access by PIF, $(k - k1)$ accesses should cross over the time 0. Moreover, assuming that the access by PIF is delayed for time $(k - 1 + x)$, $(k - k1)$ accesses before the time 0 should interfere for time $d1 = ((k - k1) - 1 + x)$. The probability of delay time d1 can be calculated by W_{FCFS}.

There can be two cases where the access by PIF is delayed by $k1$ accesses issued after the time 0.

One is the situation where all $k1$ accesses are issued in time region $[0, d1]$. The probability of this case can be calculated by $(\int_0^{d1} f(t)dt)^{k1}$.

Another case is the situation where at least a single access ($k2 \in (0, k1)$) is issued in time region $[0, d1]$ and remaining #(k1-k2) accesses are successively issued in time region $[d1, d1 + k2]$. The probability of this case was calculated by recursive function $I_{FP_f}(f, n, k1, t_{pre}, t)$. I_{FP_f} is as follows.

$$I_{FP_f}(f, n, k1, t_{pre}, t) =$$

$$\sum_{k2=1}^{k1} ({}_{k1}C_{k2} \times (\int_{t_{pre}}^{t} f(t)dt)^{k1} \times I_{FP_f}(f, n-k1, k1-k2, t, t+k2))$$

By combining the model of FCFS and I_{FP_f}, the stochastic model of arbitration delay for the PE with lowest priority W_{FP_l} is defined as follows.

$$W_{FP_l}(f, n, k, t, d) =$$

$$\sum_{m=0}^{n} \sum_{l=0}^{k} (W_{FCFS}(f, m, l, t, d-m) \times I_{FP_f}(f, n-m, k-l, t))$$

CASE 2: MODEL FOR THE HIGHEST PRIORITY

The PE with the highest priority is interfered by only a single access which very crosses over the time 0. Therefore the PE with the highest priority will always be interfered by a single access in all situations where the PE with the lowest priority is interfered one and more than one accesses. Thus we defined the model for PE with the highest priority W_{FP_h} as follows.

$$W_{FP_h}(f,n,k,t,d) = \begin{cases} W_{FP_l}(f,n,0,t,d) & k=0 \\ \sum_{i=1}^{n} W_{FP_l(f,n,i,t,d)} & k=1 \\ 0 & k>1 \end{cases}$$

CASE 3: OTHER CASES

Here, we consider other cases where the priority of PIF is in the middle, neither the highest nor the lowest. In this case, p, the priority of PIE, is in $(0, n)$. The number of PEs whose priority is higher than PIE is p and the number of PEs whose priority is lower than PIE is (n-p).

In the middle priority case, the calculation of probability needs more division of cases than other cases since the number of accesses which interfere the access of PIE should be considered.

Fig. 8.8 illustrates the factors which should be considered. m, h, l, j represent the numbers of issues invoked before the time 0, those with higher priority, those with lower priority, and accesses which cross over the time 0, respectively.

In the calculation, consideration of accesses issued after the time 0 is simple. Only #(p-h) accesses by PEs with higher priorities than PIE should be considered using I_{FP_f} since accesses by PEs with lower priorities issued after the time 0 do not interfere the access by PIE. On the other hand, consideration of accesses issued before the time 0 is complicated since we should consider all of successive accesses which is related the accesses which cross over the time 0. In particular, in the FP arbitration, the order of accesses should be considered since the priority of accesses differ each other and they should be handled dis-

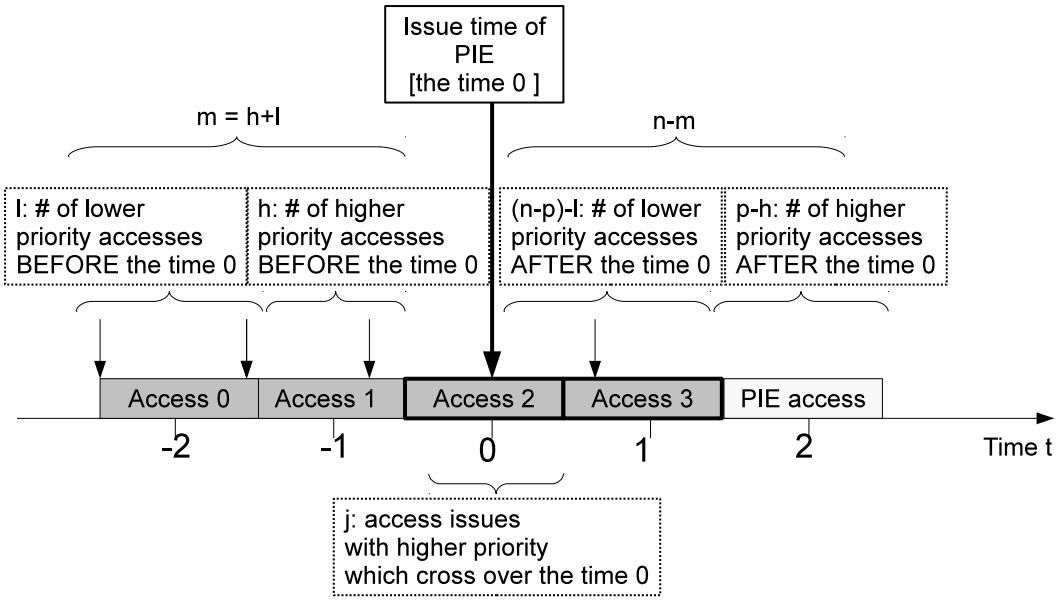


Figure 8.8: Conditions which should be considered in FP model.

tinctively. However, the consideration of accesses distinctively is not practical because of the computation cost and we abstracted out the difference of priority. Instead, we separately calculated the number of cases considering how many accesses of higher priorities are included in a case.

PURPOSE OF THE FP MODEL

With the assumptions on memory access situations described above, we constructed models which calculate arbitration delay on the situation where the number of access of each PE is one.

In arbitrations of FCFS and RR, the assumption may have little effect on the difference between the delay calculated by our models and actual delay since accesses are granted by arbitration with a certain delay in the two arbitration. However, in FP arbitration, there are cases where accesses with high priorities interfere accesses with lower priorities for long time practically. Such cases cannot be handled by our FP model.

8.3.5 THE RR MODEL

Finally, we show the model of RR $W_{RR}(f, n, k, t, d)$. There are some implementation of arbitration which called RR in practical systems since ideal round-robin arbitration is hard to implement from the point of view of costs. So we assumed the round-robin arbitration as the dynamic priority arbitration where the priority of PEs down to the lowest on each completion of their access. Based on the assumption, we constructed the following equation.

$W_{RR}(d) = CP(p = pri) \times W(p = pri, d)$, where $CP(p = pri)$ is the probability of the case where the priority of PIE is ***pri***, and $W(p = pri, d)$ is the probability of the case where the delay time become *d* when the priority p of the PIE is ***pri***. Obviously $W(p = pri, d) = W_{FP}(p = pri, d)$ holds.

In order to calculate $CP(p = pri)$, we consider the situation illustrated in Fig. 8.9. With the assumption that the probability of all PEs are $f(t) = f$, the average interval of access can be considered as $1/f$ and the time of previous access of PIE can be the time $-1/f$. The time interval $[-1/f, 0]$ is the same to the time region on which the FCFS model is constructed. In the round-robin arbitration mechanism described above, The priority of PIE become the lowest on the time $-1/f$. When j accesses completed in the time region $[-1/f, 0]$ ($j = 2$ in the figure), the j accesses became lower priority than PIE. In the situation, the priority of PIE will be $(n - j)$ at the time 0 ($n - j = 3 - 2 = 1$ in the figure). With the assumption of the FCFS model that all PEs accesses once in the time region $[-1/f, 0]$, all $(n-j)$ accesses which do not complete in $[-1/f, 0]$ can be considered to cross over the time 0. Therefore, the case of the priority of PIE is pri occurs when the number of accesses is pri considering time region $[-1/f, 0]$. This case matches the FCFS model and $CP(p = pri) = I_{FCFS}(k = pri)$ holds.

As a result, the model of RR is defined as follows.

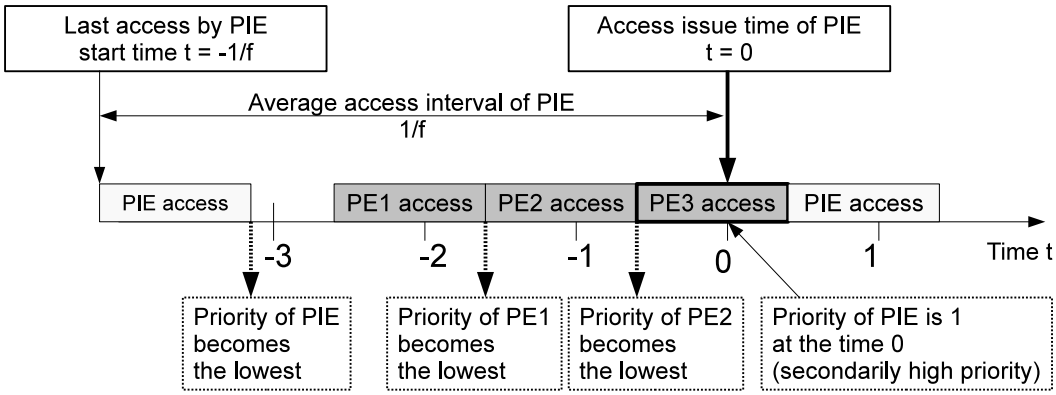


Figure 8.9: An example situation of RR model.

$$W_{RR}(f,n,k,t,d) = \sum_{pri=0}^{n} (I_{FCFS}(f,n,pri,t) \times W_{FP}(f,pri,n,k,t,d))$$

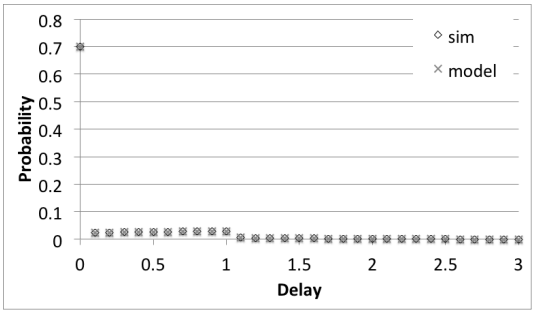
Note that this calculation is approximate one and not strict one.

8.3.6 VERIFICATION OF MODELS

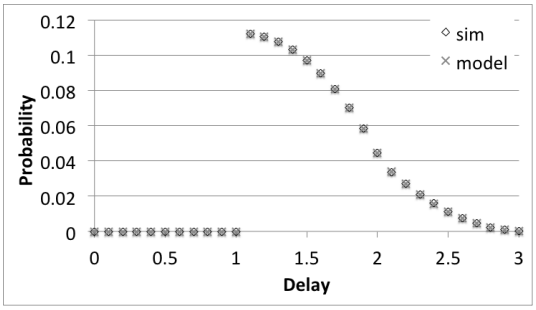
In order to verify the stochastic models, we conducted Monte Carlo simulation for comparison. Monte Carlo simulation obtains probabilistic distribution of arbitration delay on the assumed memory access situation. Note that this verification shows the validity of calculation and do not show the validity of the assumption.

Monte Carlo simulation simulates the situation where PEs accesses uniformly and randomly and stores the delay time of PIE. We compared the probabilistic distribution which obtained by executing the simulation for a certain number of time with that by our models.

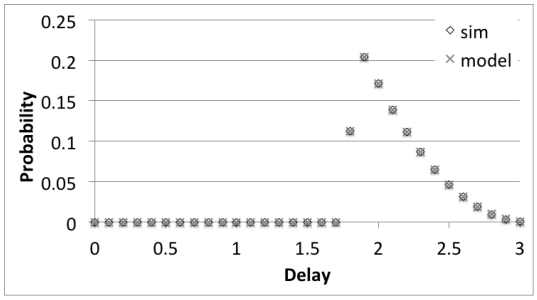
In Figs. 8.10, 8.11 and 8.12 comparison results of probabilistic distinctions for three arbitration policies. All graphs in the figure shows the result on the number of PEs are 4



(a) FCFS, f=0.1, n=3



(b) FCFS, f=0.5, n=3



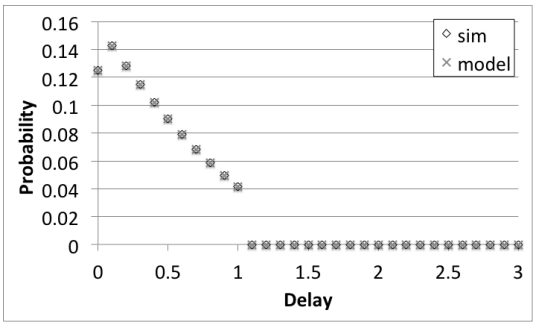
(c) FCFS, f=0.8, n=3

Figure 8.10: Probability distribution of bus arbitration delay for FCFS obtained by our models and Monte-Carlo simulation.

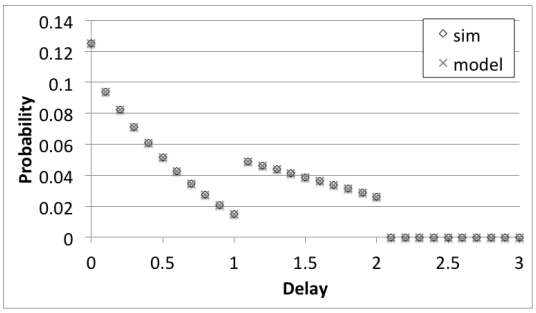
(therefore n is 3). X axis shows the delay time of PIE and Y axis shows the probabilities of the delay time. In the graphs, "sim" (denoted by rectangles) shows simulation results and "model" (denoted by crosses) shows our model results.

Comparison results about FCFS are shown in figures 8.10(a), 8.10(b) and 8.10(c) for access probabilities on 0.1, 0.5 and 0.8, respectively. In the figures, the results of simulations and models match well. The errors of expected values were under 1 % for three access probabilities.

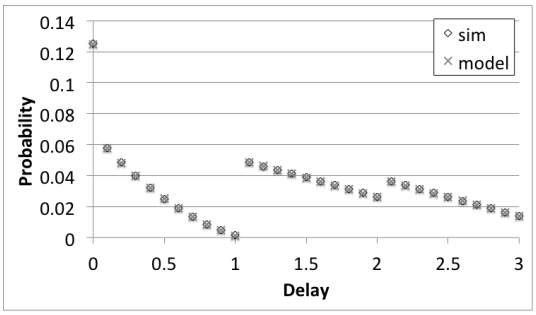
Next, comparison results about FP are shown in figures 8.11(a), 8.11(b), 8.11(c) and 8.11(d) where priorities of PIE are 0, 1, 2 and 3, respectively. Access probabilities of PEs f were set to 0.5 in all four comparisons. From the four comparisons, we could see that probabilistic distribution matched well and errors on expected values of them were under 1%. However, for low probabilities such as $f = 0.1, 0.2 or 0.3$ (not shown in the figures),



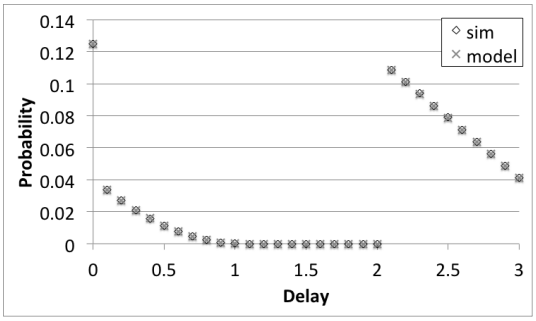
(a) FP, f=0.5, n=3, priority 0



(b) FP, f=0.5, n=3, priority 1



(c) FP, f=0.5, n=3, priority 2



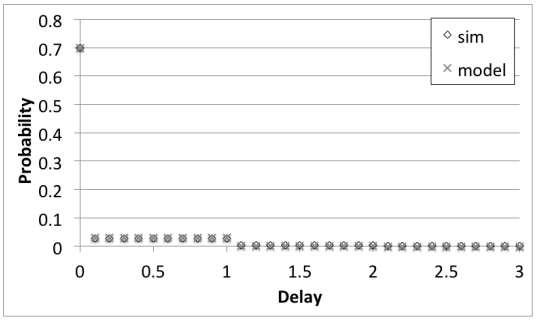
(d) FP, f=0.5, n=3, priority 3

Figure 8.11: Probability distribution of bus arbitration delay for FP obtained by our models and Monte-Carlo simulation.

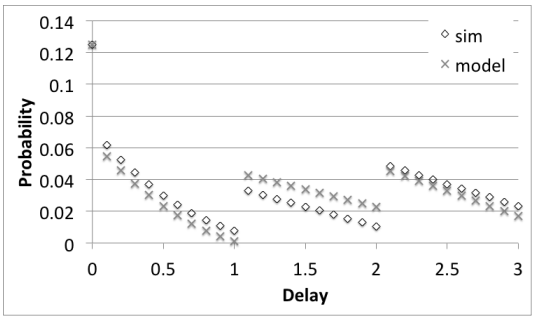
errors increased on middle priorities such as $pri = 2$ and the expected values got -4% at maximum. The reason of this increase of errors are thought to be the effect of simplification on the FP model.

Finally, we show comparison results about RR in figures 8.12(a), 8.12(b) and 8.12(c) for access probabilities of PEs on 0.1, 0.5 and 0.8, respectively. From the figures, relatively large errors could be seen. Errors on expected values were about -13% in low access probabilities and about -3% in high access probabilities. The reason of this large errors can be thought to be the effect of simplification of the RR model. However, the from of probabilistic distributions obtained by the models are similar to those by Monte Carlo simulations and therefore the RR model can be thought to be qualitatively valid.

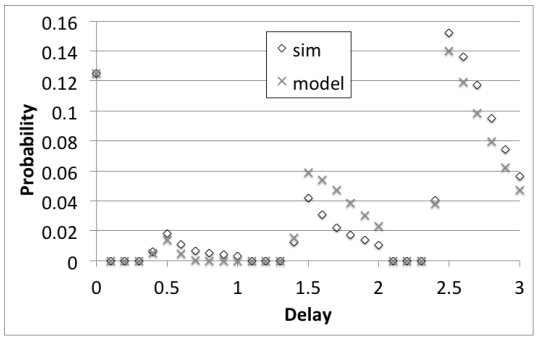
With above comparison results, the validity of models for FCFS and FP were shown. The errors of RR models were relatively large and the RR model should be improved.



(a) RR, f=0.1, n=3



(b) RR, f=0.5, n=3



(c) RR, f=0.8, n=3

Figure 8.12: Probability distribution of bus arbitration delay for RR obtained by our models and Monte-Carlo simulation.

8.4 A CASE STUDY ON JPEG DECODER SYSTEM DESIGN

In order to demonstrate the effectiveness of our performance estimation framework, we show a case study of JPEG decoder system design.

The case study was performed on the following environment. The systems were designed using SystemBuilder-MP on a PC whose OS is Windows XP Professional with an Intel Core 2 Quad 2.66 GHz processor and 2 GB RAM. The target board has an Altera Stratix II FPGA with Nios II soft-core processors at 50 MHz of clock frequency. eXCite 3.2c [46] was used for behavioral synthesis. Logic synthesis and P&R were done by Quartus 8.1. Performance estimation was performed on a PC whose OS is Linux with an Intel Xeon 2.93 GHz processor with 8 logical processor cores. SystemPerfEst is implemented in Python programming language and executed using psyco[63] which is a JIT compiler for

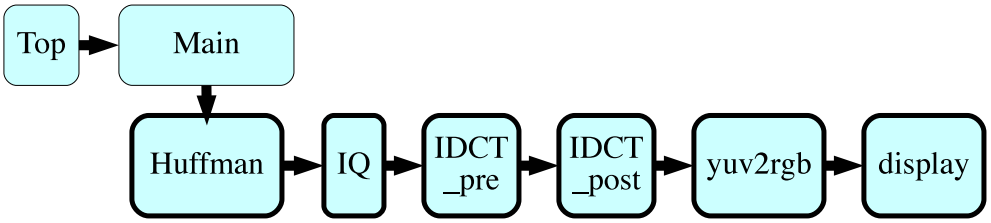


Figure 8.13: Functional structure of the JPEG decoder system.

Python.

8.4.1 JPEG DECODER DESIGN SPACE EXPLORATION

First, we designed a JPEG decoder system (Fig. 8.13). In Fig. 8.13, rectangles represent processes and processes with thick border can be mapped onto both processors and dedicated hardware. Arrows among processes represent FIFOs, which consist of FIFO channels and memory channels (as described in Chapter 4). For example, an arrow between "IQ" and "IDCT_pre" represent a FIFO which transfers 8×8 pixels of data with a buffer, Each FIFO shown in the JPEG decoder has three buffers. Process "top" and "main" can be implemented onto software only. Processes in the JPEG decoder can run in parallel in a pipelined manner. We evaluated performance of the JPEG decoder with an input image data in QVGA size (320×240 pixels).

Then we synthesized two FPGA-based prototypes of JPEG decoder (all software implementation and all hardware one) with non-preemptive scheduling policy, and obtained profiles of them using SystemBuilder-MP. Note that all memory channels used for FIFO buffers are mapped onto an on-chip SRAM and fixed in this exploration. Since the on-chip SRAM is fast, effects of memory conflicts are supposed to be negligibly small. Also, we synthesized an architecture characterizer description twice in order to obtain communication time on the FPGA with RTOSs for single processor systems (TOPPERS/JSP kernel) and for multiprocessor systems (TOPPERS/FDMP kernel).

mapping	Huffman	IQ	IDCT_pre	IDCT_post	yuv2rgb	display
2core-pipe-1	1	2	2	2	2	2
2core-pipe-2	1	1	2	2	2	2
2core-pipe-3	1	1	1	2	2	2
2core-pipe-4	1	1	1	1	2	2
2core-pipe-5	1	1	1	1	1	2
2core-zigzag	1	2	1	2	1	2
3core-pipe-1	1	2	2	**3**	**3**	**3**
3core-pipe-2	1	2	2	2	**3**	**3**
3core-pipe-3	1	2	2	2	2	**3**
3core-pipe-4	1	1	2	**3**	**3**	**3**
3core-pipe-5	1	1	2	2	**3**	**3**
3core-pipe-6	1	1	2	2	2	**3**
3core-zigzag	1	2	**3**	1	2	**3**
1core-hw-1	HW	1	HW	1	HW	1
1core-hw-2	HW	1	1	1	HW	HW
1core-hw-3	HW	1	1	1	HW	1
1core-hw-4	1	HW	HW	HW	1	1
2core-hw-1	1	2	2	2	HW	1
2core-hw-2	HW	2	2	2	1	1
2core-hw-3	1	2	2	2	HW	HW
2core-hw-4	1	1	2	2	HW	HW
2core-hw-5	HW	2	2	2	1	HW

Figure 8.14: Explored mapping of the JPEG decoder system.

After that, we explored 22 mappings of the JPEG decoder (shown in Fig. 8.14) on target architecture shown in Fig. 8.4 using both our framework and FPGA-based evaluation method with SystemBuilder-MP, and compared them. In the figure, “1”,“2”,“3” indicate processor number, and “HW” means hardware. For example, mapping “2core-hw-1” is a mapping where processes “Huffman” and “display” are mapped onto processor 1, processes “IQ”, “IDCT_pre”, “IDCT_post” are mapped onto processor 2 and process “yuv2rgb” is mapped onto hardware.

These mappings are selected in order to show that SystemPerfEst can estimate performance of mappings with various characteristics accurately. Since processes of the JPEG decoder system are executed in a parallel and a pipelined manner, typical cases of their mapping are separating them into former part (“Huffman” side part) and latter part (“display” side part), for utilizing their parallelism. Mappings of “2core-pipe-*” and “3core-

pipe-*" are such typical cases. However, these mappings have only a single or two inter-processor communications (communications between processors) at separation point. In order to show the accuracy of estimation for mappings where many inter-processor communications occur, we also show the estimation results of mappings "2core-zigzag" and "3core-zigzag". Communications between any two processes in these mappings are inter-processor communications. These two mappings are also typical cases which cannot utilize the parallelism of pipeline.

On mappings where some processes are mapped onto hardware, interrupt handling time has a large effect on their performance. Therefore, our performance estimation method should also consider the time correctly. In the mapping of "1core-hw-1", processes are mapped onto a processor and hardware alternately and therefore most of their communications are software-hardware communications which use interruption. In contrast, the mapping of "1core-hw-2" have less software-hardware communications than "1core-hw-1". Moreover, the mapping "1core-hw-3" have no hardware-hardware communication but the mapping "1core-hw-4" have two hardware communications. The mapping "2core-hw-*" shows the accuracy with two processors and hardware modules.

In summary, the 22 mappings are considered to be sufficient to demonstrate the accuracy of SystemPerfEst.

8.4.2 ACCURACY

Fig. 8.15 shows a comparison between measured execution times of FPGA-based prototypes synthesized by SystemBuilder-MP and estimated execution times. In the figure, left side bars show execution time of mappings obtained from FPGA-based prototypes, while right side bars show estimated execution time by SystemPerfEst.

Mean absolute error (MAE) of estimation results was 1.92%, distributing from -4.77% to 1.74%, which is thought to be sufficient for comparative evaluation of mappings.

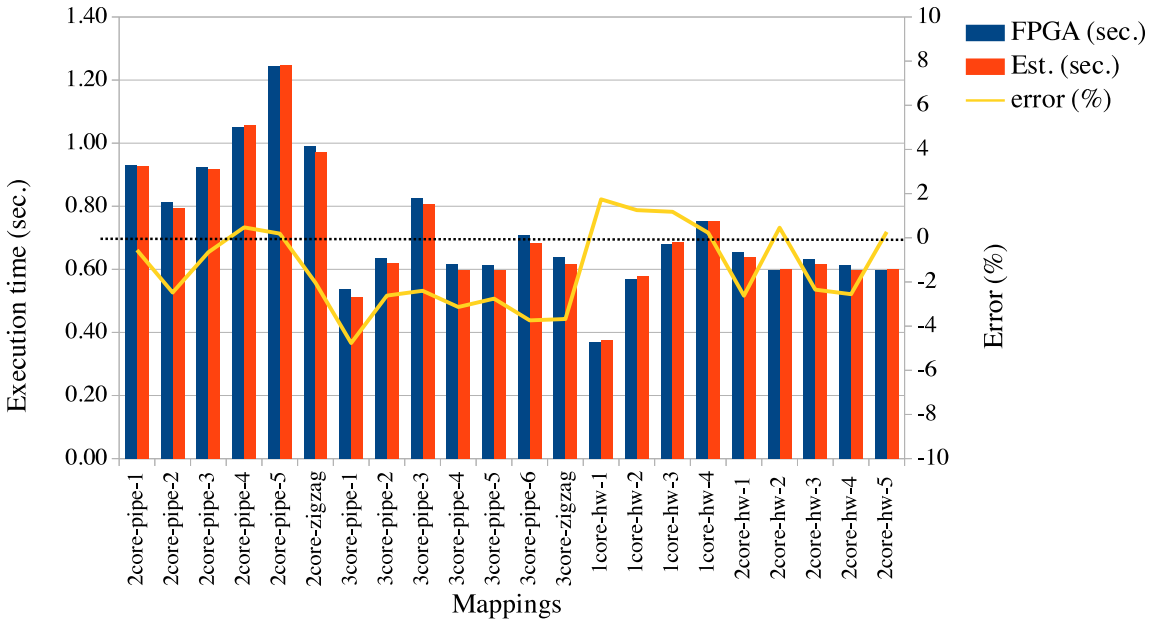


Figure 8.15: Performance estimation results of the JPEG decoder system using on-chip SRAM for FIFO buffers.

One of the reason of these good estimation results is architecture characteristics which is obtained for both RTOSs for single processor systems (single processor RTOS characteristics) and multiprocessor systems (multiprocessor RTOS characteristics), as discussed in 8.2.9. As an example, we show a comparison of the performance estimation results of 2 processor systems ("2core-pipe-*") with single processor RTOS characteristics and multiprocessor RTOS characteristics. With multiprocessor characteristics, MAE was 0.87%. In contrast, MAE was 3.86% with single processor RTOS characteristics.

There are some amount of errors in estimation results of SystemPerfEst as described above. Nevertheless, it is most important for designers to perform comparative evaluation of mappings in design space exploration. The results in Fig. 8.15 indicate that two mappings where one was superior to another over about 10 % on FPGA-based prototypes kept their inferior-to-superior relationships on estimation results. In other words, SystemPerfEst

could not differentiate mappings whose difference of performances is in about 10% in our experiment. However, it may not be crucial problem in a system-level design space exploration since the difference of such mappings with small differences in performance should be discussed with not only performance of them but also their costs such as area and power consumptions.

8.4.3 EXPLORATION TIME

With our integrated framework, we could use SystemPerfEst seamlessly without modifying functional description and profiles of FPGA-based prototypes.

SystemPerfEst took 0.9 hours for obtaining two input profiles and architecture characteristics using SystemBuilder-MP, and took about 20 seconds for the 22 mappings, 0.9 seconds per a mapping in average. In contrast, FPGA-based evaluation method took about 6.5 hours in total for the 22 mappings, 0.3 hours per a mapping in average. In detail, synthesis by SystemBuilder-MP and a behavioral synthesis tool took about 2.5 hours, logic synthesis and P&R took about 3 hours for nine mappings with dedicated hardware. Software compilation, execution of systems and recording results for 22 mappings took about 1 hour.

With measurement above, estimation time of SystemPerfEst can be formalized as $0.9 + 0.00025 \times N$ hours for N mappings. Evaluation with FPGA-based evaluation method can be formalized as $0.7 \times N$ hours for mappings with some dedicated hardware and $0.05 \times N$ hours for mappings with no dedicated hardware. For the large N, therefore, SystemPerfEst can perform exploration 2,800 times faster than FPGA-based evaluation method for mappings with some dedicated hardware, and 200 times faster even for mappings with no dedicated hardware.

8.4.4 EFFECTS OF MEMORY CONFLICTS

In order to examine effects of memory conflicts, we changed mapping of FIFO buffers from on-chip SRAM to off-chip SDRAM, and compared estimation results and evaluation results on FPGA-based prototypes of mappings "2core-pipe-*" and "3core-pipe-*" (shown in Fig. 8.16). In the results, we also showed the comparison of a mapping which all processes are mapped onto a single processor (denoted as "1core" in Fig. 8.16). The "1core" mapping have no memory conflict in the execution.

First, from results of the "1core" mapping (-0.42 % error) , we could show that SystemPerfEst can estimate performance of mappings which differ not only processes but also memories accurately.

Then we focused on the results of mappings with two and three processors, errors got worse than those in Fig. 8.15. MAE increased to 8.47%. Moreover, distribution of errors got wide, from -0.42% to -15.52%. Since processors on target architecture use no cache, the cause of these negative errors is supposed to be memory conflicts on an off-chip SDRAM. If more processors or dedicated hardware are used for more parallelism, estimation errors may get worse than the results in Fig. 8.16. Therefore techniques which can estimate effects of memory conflicts in short time are necessary, and we are currently working on this topic.

8.4.5 ESTIMATION OF ARBITRATION DELAY

As shown in 8.4.4, memory conflicts on SDRAM accesses have a serious effects on estimation error. In this evaluation, we apply our stochastic model of arbitration delay for SDRAM access conflicts on JPEG decoder system and show the effectiveness of our stochastic model.

In order to apply our stochastic models to SystemPerfEst, SystemPerfEst was modified

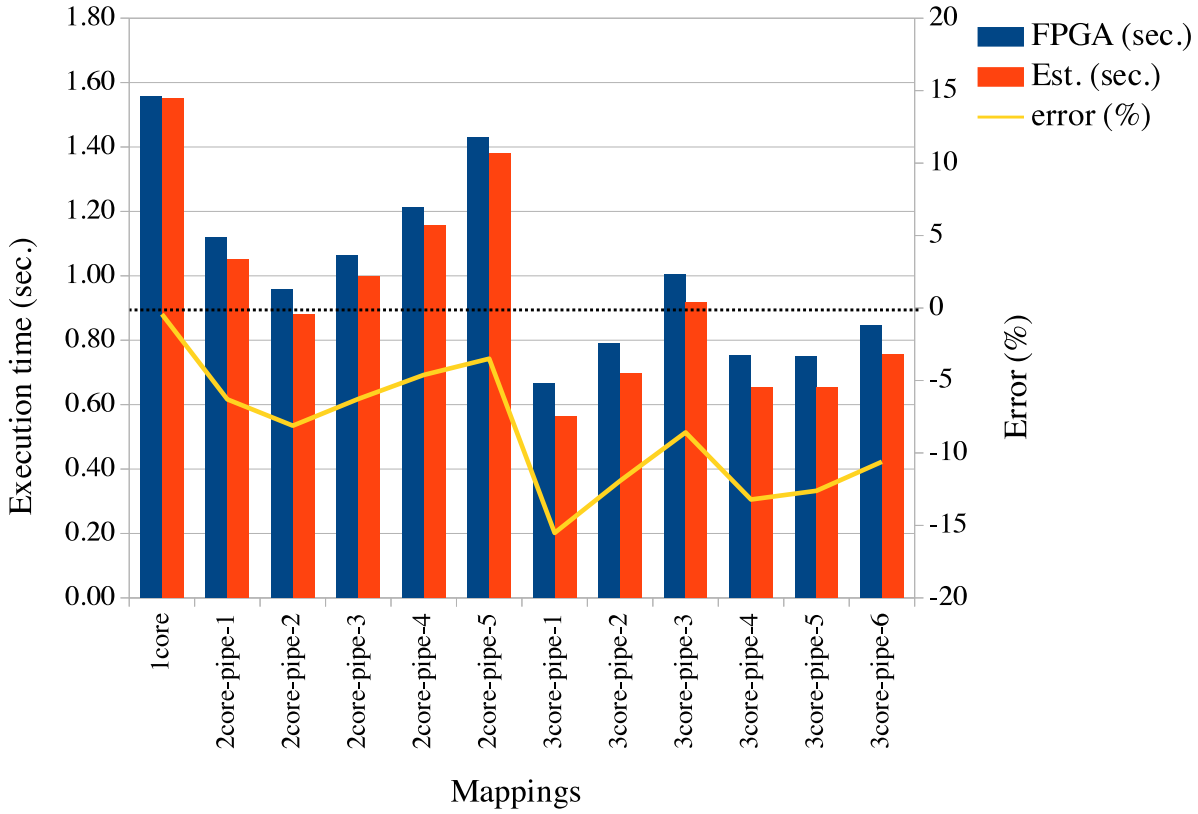


Figure 8.16: Performance estimation results of the JPEG decoder system using off-chip SDRAM for FIFO buffers.

to calculate average probability of memory access for each conflict time period. In each conflict time period, expected values of the models are used to obtain average delay for a certain access probability of processors.

Since Altera FPGA generates round-robin arbitration circuitry for arbitration on the interface of SDRAM, we applied RR-model for the JPEG decoder system.

Before the execution of performance estimation with SystemPerfEst, we first developed a data table which have expected values for memory access probabilities from 0.00 to 1.00 by 0.01 (e.g., 0.01, 0.02, . . . 1.00) since calculation of expected values takes long time.

We applied the RR model to 11 mappings of “2core-pipe-*” and “3core-pipe-*” shown

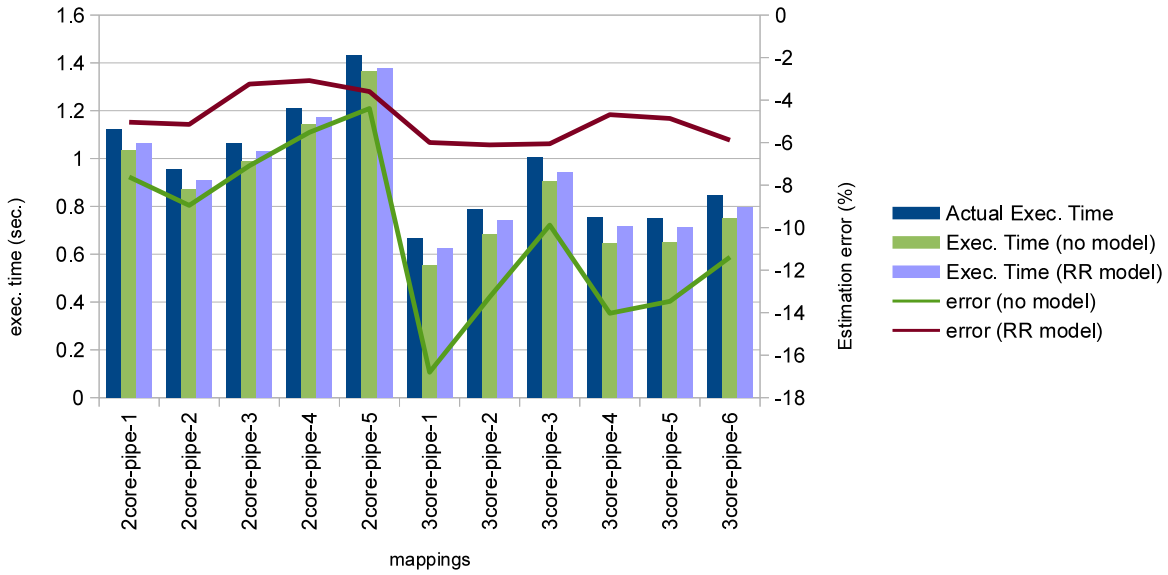


Figure 8.17: Performance estimation results of the JPEG decoder system on SDRAM with the RR model.

in Fig. 8.16. The results are shown in Fig. 8.17. In the figure, "Exec. time (no model)" and "error (no model)" shows estimated execution times and estimation errors without the RR model, respectively. "Exec. time (RR model)" and "error (RR model)" shows estimated execution time and estimation errors with the RR model. In the figure, we can see that overall estimation errors were decreased by application of the RR model. Especially, the errors on mappings with three processors were largely decreased. As a result, average error of estimation results decreased from -10.2% to -4.9 %. Moreover, distribution of errors were also improved from $[-16.8\%, -4.4\%]$ to $[-6.1\%, -3.1\%]$.

8.5 CONCLUSIONS

In system-level design, system designers describe functionalities as processes and channels, and iterates mapping of processes onto processing elements and evaluation. We proposed

a fast performance estimation framework for system-level design exploration, combining SystemBuilder-MP and a newly developed trace-based simulation tool, named SystemPerfEst.

Since SystemPerfEst works in close cooperation with SystemBuilder-MP, designers easily estimate performance of design candidates exhaustively after describing functionalities of a system. Moreover, with the architecture characterizer description provided by our estimation framework, designers easily reflect characteristics of target FPGAs, memory modules and RTOSs.

In order to estimate performance of systems considering arbitration delay on simultaneous memory accesses, we proposed stochastic models of bus arbitration delay on three arbitration policies: FCFS, Fixed-Priority and Round-Robin.

We demonstrated the effectiveness of our performance estimation method through a case study on design space exploration of a JPEG decoder system. In design space exploration of the JPEG decoder system, performance estimation results of SystemPerfEst achieved 1.92% in mean average error on systems with a fast memory module. Also we achieved -5.1% average error on systems with a slow memory module by a stochastic model of Round-Robin arbitration delay.

As a future work, consideration of caches on processors should be added. Moreover, we are developing an efficient design space exploration strategy and an exploration automation tool which uses SystemPerfEst according to the strategy.

CHAPTER 9

CONCLUSIONS

This dissertation presented system-level design framework. Our framework consists of four tools: SystemBuilder-MP, covalidation environment, system-level profilers and a fast performance estimation tool.

In Chapter 4, we explained a base tool SystemBuilder and cosimulation environment, and showed an case study on MPEG-4 decoder design. SystemBuilder synthesize the implementation of a system with a single processor and a dedicated hardware module from functional description of the system. In the case study, sequential description of MPEG-4 decoder was divided into processes with pipelined parallel structure.

In Chapter 5, we proposed SystemBuilder-MP. SystemBuilder-MP is an automatic synthesis tool which synthesize implementation of systems on multiprocessor architecture from system description with processes and channels. In order to deal with multiprocessor architecture, SystemBuilder-MP provides channels for mutual exclusion and ring buffer management, and utilizes an RTOS for multiprocessor. Moreover, SystemBuilder-MP synthesizes multiple interface ports for dedicated hardware in order to enable dedicated hardware access and to be accessed without conflicts on the ports.

In Chapter 6, our covaliation environment is proposed for debug of functionalities of a

system. The covalidation environment utilizes an RTOS-model for fast software execution and an FPGA for fast hardware execution.

In Chapter 7 proposed system-level profilers used in FPGA-based prototypes generated by SytemBuilder-MP. The profilers consists of a process profiler and a memory profiler. The process profiler records activation and wait timings of processes and the memory profiler records memory accesses from a dedicated hardware module. The effect of the process profiler is demonstrated in a case study on multiprocessor implementation of AES encryption system design and the effect of the memory profiler is demonstrated in a case study of MPEG-4 decoder system design. In the case study of MPEG-4 decoder design, the process profiler and the memory profiler contributed to find bottlenecks of systems and to improve them.

In order to make design space exploration more efficient, Chapter 8 proposed a fast performance estimation tool. With the fast performance estimation method, system designer only needs synthesize a few FPGA-based prototype implementations of a system and performance of other mappings are estimated on a host PC. Stochastic models of bus arbitration delay are also proposed in order to estimate performance of systems which involve memory access conflicts in their execution. With the performance estimation, exploration of JPEG decoder design was performed seven times faster than exploration with SystemBuilder-MP only, and achieved small error within about 6%.

As shown in the case study on AES encryption, JPEG decoder and MPEG-4 decoder design in the chapters, system designers can explore design space at system level efficiently with our framework constructed by above tools.

ACKNOWLEDGEMENT

I am profoundly indebted to my adviser, Associate Professor Shinya Honda, for his guidance, understanding, generosity, and most importantly, his sincerity during my studies at Nagoya University.

I would like to deeply express my respect and gratitude to Professor Hiroaki Takada and Professor Hiroyuki Tomiyama (currently with Ritsumeikan University) for giving invaluable suggestions and directions throughout the work. Their insightful indications and comments on the work led me to foster an ability of seeing things from different standpoints. It was my great pleasure to be given numerous opportunities to work in the fields of system-level design of embedded systems and to have studied in Takada Laboratory for six years.

This research was in part supported by STARC (Semiconductor Technology Academic Research Center) as collaborative research projects from 2006 to 2008 and from 2009 to 2011. I would like to thank researchers and guest researchers of STARC for helpful discussions and advice.

I am very grateful to Professor Masato Edahiro for serving on a committee member of this dissertation and providing invaluable suggestions and comments.

I would like to express my special thank to the past and present members of Takada Laboratory. Their helpful comments and cooperation were encouraging and stimulating to me, and fulfilling discussions with them led me to have new ideas on my studies. Also,

thanks to them, my academic life was very enjoyable.

I would like to thank YXI and Soliton for providing a high-level synthesis tool and technical supports.

Last but not least, I thank my family for their never-ending support, encouragements, and their belief in me.

My financial support for this work has been in part provided by the Research Fellowship of the Japan Society for the Promotion of Science for Young Scientists.

BIBLIOGRAPHY

[1] A. Jerraya and W. Wolf, "Hardware/software interface codesign for embedded systems," *Computer*, vol. 38, no. 2, pp. 63 – 69, feb. 2005.

[2] A. Kalavade and E. A. Lee, "Hardware/software co-design using ptolemy - a case study," in *IFIP International Workshop on Hardware/Software Co-Design*, May 1992.

[3] W. H. Wolf, "Hardware-software co-design of embedded systems," *Proceedings of the IEEE*, vol. 82, no. 7, pp. 967 –989, jul 1994.

[4] G. De Michell and R. Gupta, "Hardware/software co-design," *Proceedings of the IEEE*, vol. 85, no. 3, pp. 349 –365, mar 1997.

[5] I. Harris, "Hardware/software covalidation," *Computers and Digital Techniques, IEE Proceedings* -, vol. 152, no. 3, pp. 380 – 392, may 2005.

[6] W. Wolf, A. Jerraya, and G. Martin, "Multiprocessor system-on-chip (MPSoC) technology," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 10, pp. 1701–1713, Oct. 2008.

[7] K. S. Chatha and R. Vemuri, "An iterative algorithm for hardware-software partitioning, hardware design space exploration and scheduling," *Design Automation for Embedded Systems*, vol. 5, pp. 281–293, 2000.

[8] V. Srinivasan, S. Govindarajan, and R. Vemuri, "Fine-grained and coarse-grained behavioral partitioning with effective utilization of memory and design space exploration for multi-fpga architectures," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 9, no. 1, pp. 140 –158, feb. 2001.

[9] A. Baghdadi, N. Zergainoh, W. Cesario, T. Roudier, and A. Jerraya, "Design space exploration for hardware/software codesign of multiprocessor systems," in *Rapid System Prototyping, 2000. RSP 2000. Proceedings. 11th International Workshop on*, 2000, pp. 8 –13.

[10] T. Givargis and F. Vahid, "Platune: a tuning framework for system-on-a-chip platforms," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 21, no. 11, pp. 1317 – 1327, nov 2002.

[11] R. Dömer, A. Gerstlauer, J. Peng, D. Shin, L. Cai, H. Yu, S. Abdi, and D. D. Gajski, "System-on-chip environment: a SpecC-based framework for heterogeneous MPSoC design," *EURASIP Journal on Embedded Systems*, vol. 2008, pp. 1–13, 2008.

[12] M. Thompson, H. Nikolov, T. Stefanov, A. D. Pimentel, C. Erbas, S. Polstra, and E. F. Deprettere, "A framework for rapid system-level exploration, synthesis, and programming of multimedia MP-SoCs," in *CODES+ISSS*, 2007, pp. 9–14.

[13] S. Ha, S. Kim, C. Lee, Y. Yi, S. Kwon, and Y.-P. Joo, "PeaCE: A hardware-software codesign environment for multimedia embedded systems," *ACM Trans. on Design Automation of Electronic Systems*, vol. 12, no. 3, pp. 1–25, 2007.

[14] J. Keinert, M. Streubühr, T. Schlichter, J. Falk, J. Gladigau, C. Haubelt, J. Teich, and M. Meredith, "Systemcodesigner –an automatic esl synthesis approach by design space exploration and behavioral synthesis for streaming applications," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 14, no. 1, pp. 1–23, 2009.

[15] A. Gerstlauer, C. Haubelt, A. Pimentel, T. Stefanov, D. Gajski, and J. Teich, "Electronic system-level synthesis methodologies," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 10, pp. 1517–1530, Oct. 2009.

[16] D. Gajski, J. Zhu, R. Dömer, G. A., and S. Zhao, *SpecC: Specification Language and Methodology*. Kluwer Academic Publishers, 2000.

[17] "SystemC Open Initiative," http://www.systemc.org/.

[18] Y. Hwang, S. Abdi, and D. Gajski, "Cycle-approximate retargetable performance estimation at the transaction level," in *DATE '08: Proceedings of the conference on Design, automation and test in Europe*. New York, NY, USA: ACM, 2008, pp. 3–8.

[19] H. Hihara, S. Moriyama, T. Takezawa, Y. Nishihara, M. Nomachi, T. Takahashi, and T. Takashima, "Designing space cube 2 with ELEGANT framework," in *International SpaceWire Conference*, Nov. 2008.

[20] A. Gerstlauer, J. Peng, D. Shin, D. Gajski, A. Nakamura, D. Araki, and Y. Nishihara, "Specify-explore-refine (SER): from specification to implementation," in *the 45th annual Design Automation Conference (DAC)*. New York, NY, USA: ACM, 2008, pp. 586–591.

[21] K. Wakabayashi and T. Okamoto, "C-based SoC design flow and EDA tools: an ASIC and system vendor perspective," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 12, pp. 1507–1522, Dec 2000.

[22] D. Pellerin and S. Thibault, "Practical FPGA programming in C," *Prentice Hall Press*, 2005.

[23] A. Gerstlauer, H. Yu, and D. D. Gajski, "RTOS modeling for system level design," in *Design, Automation and Test in Europe (DATE)*. Washington, DC, USA: IEEE Computer Society, 2003, p. 10130.

[24] Y. Yi, D. Kim, and S. Ha, "Virtual synchronization technique with os modeling for fast and time-accurate cosimulation," in *the 1st IEEE/ACM/IFIP International Conference on Hardware/software Codesign and System Synthesis (CODES+ISSS)*, 2003.

[25] L. Formaggio, F. Fummi, and G. Pravadelli, "A timing-accurate hw/sw co-simulation of an iss with systemc," in *CODES+ISSS*, September 2004.

[26] M. Krause, D. Englert, O. Bringmann, and W. Rosenstiel, "Combination of instruction set simulation and abstract rtos model execution for fast and accurate target software evaluation," in *Proceedings of the 6th IEEE/ACM/IFIP international conference on Hardware/Software codesign and system synthesis*. New York, NY, USA: ACM, 2008, pp. 143–148.

[27] Y. Nakamura, K. Hosokawa, I. Kuroda, K. Yoshikawa, and T. Yoshimura, "A fast hardware/software co-verification method for system-on-a-chip by using a c/c++ simulator and fpga emulator with shared register communication," in *Design Automation Conference (DAC)*, 2004.

[28] "Xilinx,"
http://www.xilinx.com/tools/cspro.htm.

[29] "Altera Corporation, Signal-Tap,"
http://www.altera.com/support/examples/on-chip-debugging/signal-tap/signaltap2-design-examples.html.

[30] P. G. D. Valle, D. Atienza, I. Magan, J. G. Flores, E. A. Perez, J. M. Mendias, L. Benini, and G. D. Micheli, "A complete multi-processor system-on-chip FPGA-based emulation framework," in *IFIP VLSI-SoC*, Oct. 2006.

[31] D. Nunes, M. Saldaa, and P. Chow, "A profiler for a heterogeneous multi-core multi-fpga system," in *ICFPT*, December 2008.

[32] M. Nanjundappa, H. D. Patel, B. A. Jose, and S. K. Shukla, "Scgpsim: a fast systemc simulator on gpus," in *Proceedings of the 2010 Asia and South Pacific Design Automation Conference*. Piscataway, NJ, USA: IEEE Press, 2010, pp. 149–154.

[33] T. Isshiki, D. Li, H. Kunieda, T. Isomura, and K. Satou, "Trace-driven workload simulation method for multiprocessor system-on-chips," in *the 46th Design Automation Conference (DAC)*. ACM, 2009, pp. 232–237.

[34] A. Bouchhima, P. Gerin, and F. Pétrot, "Automatic instrumentation of embedded software for high level hardware/software co-simulation," in *Proceedings of the 2009 Asia and South Pacific Design Automation Conference*. Piscataway, NJ, USA: IEEE Press, 2009, pp. 546–551.

[35] S. Mahadevan, K. Virk, and J. Madsen, "ARTS: A systemc-based framework for multiprocessor systems-on-chip modelling," in *Design Automation for Embedded Systems*, 2007.

[36] T. Wild, A. Herkersdorf, and G.-Y. Lee, "TAPES - trace-based architecture performance evaluation with SystemC," *Design Automation for Embedded Systems*, vol. 10, no. 2-3, pp. 157–179, Sep. 2005.

[37] L. Cai, A. Gerstlauer, and D. Gajski, "Retargetable profiling for rapid, early system-level design space exploration," in *DAC*, June 2004.

[38] A. D. Pimentel, M. Thompson, S. Polstra, and C. Erbas, "Calibration of abstract performance models for system-level design space exploration," *Journal of Signal Processing Systems*, vol. 50, no. 2, pp. 99–114, 2008.

[39] K. Ueda, K. Sakanushi, Y. Takeuchi, and M. Imai, "Architecture-level performance estimation method based on system-level profiling." *IEE Proceedings – Computers & Digital Techniques*, vol. 152, no. 1, pp. 12 – 19, 2005.

[40] "Carbon Design Systems, Inc."
http://www.carbondesignsystems.co.jp/.

[41] "CoWare Inc."
http://www.coware.com/.

[42] A. Bobrek, J. Paul, and D. Thomas, "Stochastic contention level simulation for single-chip heterogeneous multiprocessors," *Computers, IEEE Transactions on*, vol. 59, no. 10, pp. 1402 –1418, 2010.

[43] S. Honda, H. Tomiyama, and H. Takada, "Systembuilder: A system level design environment (in japanese)," *IEICE Trans. D*, vol. J88-D-I, no. 2, pp. 163–174, Feb. 2005.

[44] T. Furukawa, S. Honda, H. Tomiyama, and H. Takada, "A hardware/software cosimulator with RTOS supports for multiprocessor embedded systems," in *Third International Conferences on Embedded Software and Systems (ICESS)*, Daegu, Korea, May 2007, pp. 283–294.

[45] "TOPPERS/JSP kernel,"
http://www.toppers.jp/en/jsp-kernel.html.

[46] "Y Explorations, Inc. eXCite,"
http://www.yxi.com/index.html.

[47] "Altera Corporation,"
http://www.altera.com/.

[48] S. Honda, T. Wakabayashi, H. Tomiyama, and H. Takada, "RTOS-centric cosimulator for embedded system design," *IEICE Trans. Fundamentals*, vol. E87-A, no. 12, Dec. 2004.

[49] S. Chikada, H. Tomiyama, S. Honda, and H. Takada, "Cosimulation of ITRON-Based embedded software with SystemC," in *International High Level Design Validation and Test Workshop (HLDVT)*, Nov. 2005, pp. 71–76.

[50] "ITRON project,"
http://ertl.jp/ITRON/home-e.html.

[51] "EEMBC – The Embedded Microprocessor Benchmark Consortium,"
http://www.eembc.org/.

[52] "Toshiba: an outline of MeP (in Japanese),"
http://www.semicon.toshiba.co.jp/docget.jsp?dnum=BCJ0043&type=catalog.

[53] "Renesas Electronics EMMA Mobile^{TM} EV series (in japanese),"
http://www2.renesas.com/mobile/ja/emma_mobile/em_ev.html.

[54] H. Takada and K. Sakamura, "Towards a scalable real-time kernel for function-distributed multiprocessors," in *20th IFAC/IFIP Workshop on Real-Time Programming*, 1995.

[55] S. Honda, H. Tomiyama, and H. Takada, "RTOS and codesign toolkit for multiprocessor systems-on-chip," in *12th Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan. 2007.

[56] M. A. Hassan, K. Sakanushi, Y. Takeuchi, and M. Imai, "Rtk-spec tron: A simulation model of an itron based rtos kernel in systemc," in *Design, Automation and Test in Europe (DATE)*, 2005.

[57] H. Posadas, E. Villar, and F. Blasco, "Real-time operating system modeling in systemc for hw/sw co-simulation," in *Design of Circuits and Integrated Systems (DCIS)*, 2005.

[58] Y. Yi, D. Kim, and S. Ha, "Fast and accurate cosimulation of MPSoC using trace-driven virtual synchronization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 12, December 2007.

[59] F. Fummi, M. Loghi, M. Poncino, and G. Pravadelli, "A cosimulation methodology for hw/sw validation and performance estimation," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 14, no. 2, pp. 1–32, 2009.

[60] "GTKWave," http://intranet.cs.man.ac.uk/apt/projects/tools/gtkwave/.

[61] S. Shibata, S. Honda, H. Tomiyama, and H. Takada, "A case study of MPEG4 decoder design with SystemBuilder," in *2009 International Symposium on VLSI Design, Automation & Test (VLSI-DAT)*, April 2009, pp. 355–358.

[62] "TOPPERS Project," http://www.toppers.jp/.

[63] "Psyco," http://psyco.sourceforge.net/.

[64] "Maplesoft *maple*."

List of Publications by the Author

Research Achievements Related to the Dissertation

Journal Papers

1. Seiya Shibata, Shinya Honda, Yuko Hara, Hiroyuki Tomiyama and Hiroaki Takada, “Embedded System Covalidation with RTOS Model and FPGA”, IPSJ Transactions on System LSI Design Methodology, Vol.1, pp. 126-130, Aug 2008. (short paper)

2. Seiya Shibata, Yuki Ando, Shinya Honda, Hiroyuki Tomiyama and Hiroaki Takada, “Efficient Design Space Exploration at System Level with Automatic Profiler Instrumentation”, IPSJ Transactions on System LSI Design Methodology, Vol.3, pp.179-193, Aug 2010.

3. 柴田誠也，本田晋也，冨山宏之，高田広章，”マルチプロセッサ対応システムレベル設計環境 SystemBuilder-MP”, 電子情報通信学会論文誌 D, Vol.J94-D, No.4, pp.657-670, Apr 2011.

4. Seiya Shibata, Yuki Ando, Shinya Honda, Hiroyuki Tomiyama and Hiroaki Takada, “A Fast Performance Estimation Framework for System-Level Design Space Exploration”, IPSJ Transactions on System LSI Design Methodology, Vol.5, Feb 2012 (to

appear).

INTERNATIONAL CONFERENCES

1. Seiya Shibata, Shinya Honda, Yuko Hara, Hiroyuki Tomiyama and Hiroaki Takada, “Hardware/Software Covalidation with FPGA and RTOS Model”, Proceedings of the Workshop on Synthesis And System Integration of Mixed Information Technologies (SASIMI) 2007, pp. 488-494, Oct 2007.

2. Seiya Shibata, Shinya Honda, Hiroyuki Tomiyama and Hiroaki Takada, “A Case Study on MPEG4 Decoder Design with SystemBuilder”, 2009 International Symposium on VLSI Design, Automation and Test (VLSI-DAT), pp. 355-358, Apr 2009.

3. Seiya Shibata, Yuki Ando, Shinya Honda, Hiroyuki Tomiyama and Hiroaki Takada, “Automatic Instrumentation of Prolers for FPGA-based Design Space Exploration”, 2009 International Conference on Field-Programmable Technology (FPT ’09), pp. 292-295, Dec. 2009.

4. Seiya Shibata, Shinya Honda, Hiroyuki Tomiyama and Hiroaki Takada, “Advanced SystemBuilder: A Tool Set for Multiprocessor Design Space Exploration”, Proceeding of International SoC Design Conference (ISOCC), Vol.2010, pp. 79-82, Nov 2010.

DOMESTIC SYMPOSIUMS

1. 柴田誠也, 本田晋也, 冨山宏之, 高田広章, “動作合成とFPGAを利用したCベース協調設計・検証手法”, 情報処理学会研究報告 組込みシステム, Vol.2007,No.4, Jan 2007.

2. 柴田誠也，本田晋也，冨山宏之，高田広章，“RTOS シミュレータと FPGA を用いた組込みシステムの検証環境”, DA シンポジウム 2007 論文集, pp. 151-156, Aug 2007.

3. 柴田誠也，本田晋也，冨山宏之，高田広章，“システムレベル設計環境 SystemBuilder を用いた MPEG4 デコーダの設計事例”, 電子情報通信学会技術研究報告 [VLSI 設計技術], Vol.107,No.507, pp. 43-48, Mar 2008.

4. 柴田誠也，本田晋也，冨山宏之，高田広章，“設計探索を効率化するためのシステムレベル通信モデルと自動合成ツール”, DA シンポジウム 2009, pp. 103-108, Aug 2009.

5. 柴田誠也，本田晋也，冨山宏之，高田広章，“システムレベル設計探索のための高速性能見積もり手法”, DA シンポジウム 2010, pp. 189-194, Sep 2010.

6. 柴田誠也，石川拓也，本田晋也，冨山宏之，高田広章，“バス調停の遅延時間見積もりのための確率的数学モデル”, 情報処理学会研究報告, Mar 2011.

7. 柴田誠也，石川拓也，本田晋也，冨山宏之，高田広章，“上流設計におけるバス調停遅延時間の見積もりのための確率的数学モデル”, DA シンポジウム 2011, pp. 159-164, Sep 2011.

8. 柴田誠也，安藤友樹，本田晋也，冨山宏之，高田広章，“マルチプロセッサ対応システムレベル設計環境 SystemBuilder を用いた FPGA 向け設計事例”, RECONF 研究会, Vol.111,No.218, pp. 57-62, Sep 2011.

OTHER RESEARCH ACHIEVEMENTS

JOURNAL PAPERS

1. 安積卓也，古川貴士，相庭裕史，柴田誠也，本田晋也，冨山宏之，高田広章，“オープンソース組込みシステム向けシミュレータのマルチプロセッサ拡張, コンピュータソフトウェア”, Vol.27, No.4, pp. 24-42, Nov 2010.

2. 石川拓也，安積卓也，一場利幸，柴田誠也，本田晋也，高田広章，TECS 仕様に基づいた NXT 用ソフトウェアプラットフォームの開発，コンピュータソフトウェア，2011.（採録決定）

3. Yuki ANDO, Seiya SHIBATA, Shinya HONDA, Hiroyuki TOMIYAMA and Hiroaki TAKADA,“Automatic Communication Synthesis with Hardware Sharing for Multi-Processor SoC Design”, IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences, Vol.E93-A, No.12, pp. 2509-2516, Dec 2010.

INTERNATIONAL CONFERENCES

1. Yuki ANDO, Seiya SHIBATA, Shinya HONDA, Hiroyuki TOMIYAMA and Hiroaki TAKADA,“ A Case Study on AES Encryption System Design with SystemBuilder ”, Proceedings of the Workshop on Synthesis And System Integration of Mixed Information Technologies (SASIMI) 2009, pp. 283-288, Mar 2009.

2. Yuki Ando, Seiya Shibata, Shinya Honda, Hiroyuki Tomiyama and Hiroaki Takada, “Automatic Communication Synthesis with Hardware Sharing for Design Space Exploration”, IEEE International Symposium on Circuits and Systems, pp. 1863-1866, May 2010.

DOMESTIC SYMPOSIUMS

1. 本田晋也，柴田誠也，冨山宏之，高田広章，“システムレベル設計向けプロファイラ”, 情報処理学会研究報告 組込みシステム, Vol.2008,No.7, pp. 11-16, Jan 2008.

2. 古川貴士，柴田誠也，本田晋也，冨山宏之，高田広章，“マルチプロセッサ RTOS 対応コシミュレータ”, 情報処理学会研究報告 組込みシステム, Vol.2008,No.7, pp. 17-22, Jan 2008.

3. 相庭裕史，柴田誠也，古川貴士，本田晋也，冨山宏之，高田広章，“マルチプロセッサ RTOS 対応シミュレーション環境の機能拡張と効率化”, 電子情報通信学会技術研究報告 [コンピュータシステム], Vol.107,No.558, pp. 13-18, Mar 2008.

4. 安藤友樹，柴田誠也，本田晋也，冨山宏之，高田広章，“システムレベル設計環境 SystemBuilder を用いた AES 暗号化システムの設計事例”, 電子情報通信学会 2008 年ソサイエティ大会, No.AS-1-3, Sep 2008.

5. 安藤友樹, 柴田誠也, 本田晋也, 冨山宏之，高田広章，“設計空間探索におけるハードウェア共有用通信の自動合成”, 情報処理学会　組込みシステム研究会　第 15 回研究発表会, Jan 2010.

6. 石川拓也，安積卓也，一場利幸，柴田誠也，高田広章，“UML モデルの C 言語実装における TECS の適用事例”, 情報処理学会 第 78 回プログラミング研究会, Mar 2010.

AWARDS AND HONORS

1. Grant-in-Aid for JSPS Fellows (DC1), from Apr., 2009 to 2011.

APPENDIX A

APIS PROVIDED BY SYSTEMBUILDER-MP

Here, we explain all APIs provided by SystemBuilder-MP for five channels: blocking channels (BCs), non-blocking channels (NBCs), memory channels (MEMs), exclusive control channel (EXCs) and ring buffer control channels (RBCs). In the descriptions below, `xxx` represents the name of channel.

A.1 BC

- `xxx_READ(*v)` read a single value from top of the fifo buffer in the BC channel xxx to the variable `v`. `v` is a pointer. If the fifo buffer in the BC channel xxx is empty, the process called this API will be blocked and wait until data is written by other processes. The number of data in the fifo buffer can be configured by system designers.
- `xxx_PREAD(*v)` is same as `xxx_READ` but use polling method to wait.
- `xxx_WRITE(v)` write a single value in the variable `v` to the bottom of the fifo in

the BC channel. If the fifo buffer in the BC channel xxx is full, the process called this API will be blocked and wait until data is read by other processes. The number of data in the fifo buffer can be configured by system designers.

- `xxx_PWRITE(v)` is same as `xxx_WRITE` but use polling method to wait.

A.2 NBC

- `xxx_READ(*v)` read a single value from the buffer of the NBC channel to the variable `v`.

- `xxx_WRITE(v)` write a single value in the variable `v` to the buffer of the NBC channel.

A.3 MEM

- `xxx_READ(index, *v)` read a single value in the buffer at the place indicated by index to the variable `v`.

- `xxx_WRITE(index, v)` write a single value in the variable `v` to the buffer at the place indicated by `index`.

A.4 EXCOBJ

- `xxx_LOCK()` acquire the lock of EXCOBJ xxx. If the lock is acquired by other processes, the process called this API will be blocked until the lock is released.

- `xxx_UNLOCK()` release the lock of EXCOBJ xxx.

A.5 RBCOBJ

- to write data
  - `BOOL xxx_ACQURIE_ROOM(int num)` acquire the right to mutual access to #`num` of buffers controlled by RBCOBJ xxx to write. If #num of buffers can not be obtained, the process called this API will be blocked until #num buffers are released.
  - `BOOL xxx_TACQUIRE_ROOM(int num, int timeout)` is same as `xxx_ACQUIRE_ROOM` but have time limit on blocking specified by `timeout`.
  - `BOOL xxx_PACQUIRE_ROOM(int num)` is same as `xxx_ACQUIRE_ROOM` but use polling method to wait.
  - `BOOL xxx_SINGLE_STORE(data, int offset)` write a single value `data` to the buffer indicated by `offset` in the buffers acquired by `xxx_ACQUIRE_ROOM`.
  - `BOOL xxx_ARRAY_STORE(*vector, int offset, int count)` write #`count` values in array `vector` to the buffers acquired by `xxx_ACQRUIRE_DATA`.
  - `BOOL xxx_GET_ROOM_POINTER(**ppdata)` get the pointer to the buffers acquired by `xxx_ACQUIRE_ROOM` to `ppdata`
  - `BOOL xxx_RELEASE_DATA(int num)` release #num of buffers.
- to read data
  - `BOOL xxx_ACQURIE_DATA(int num)` acquire the right to mutual access to #`num` of buffers controlled by RBCOBJ xxx to read. If #num of buffers can not be obtained, the process called this API will be blocked until #num buffers are released.

- `BOOL xxx_TACQUIRE_DATA(int num, int timeout)` is same as `xxx_ACQUIRE_DATA` but have time limit on blocking specified by `timeout`.
- `BOOL xxx_PACQUIRE_DATA(int num)` is same as `xxx_ACQUIRE_DATA` but use polling method to wait.
- `BOOL xxx_SINGLE_LOAD(*data, int offset)` read a single value from the buffers acquired by `xxx_ACQUIRE_DATA` to the variable `data`. the offset in the buffers are indicated by `offset`.
- `BOOL xxx_ARRAY_LOAD(*vector, int offset, int count)` read #`count` values from the buffers acquired by `xxx_ACQRUIRE_DATA` to array `vector`. Start point of read is indicated by `index`.
- `BOOL xxx_GET_DATA_POINTER(**ppdata)` get the pointer to the buffers acquired by `xxx_ACQUIRE_DATA` to `ppdata`.
- `BOOL xxx_RELEASE_ROOM(int num)` release #`num` of buffers.

APPENDIX B

STOCHASTIC MODELS OF BUS ARBITRATION DELAY

Here, we provide full definition of stochastic models of bus arbitration delay. The definitions below are described in the Maple language which is a programming lanuage of mathematical tool Maple [64]. The definitions can be fed to Maple and executed just as it is.

B.1 PRELIMINARY

Here, basic functions which represent access probabilities of processors are defined.

```
#------- begin: probablistic distribution function of processors --------------#
# region [-1/prob, 0] ver.
# @prob: probability (access frequency)
# @x   : time
fregionpast := prob -> x -> piecewise(-1/prob <= x and x <= 0, prob, 0);

# region [-1/(2*prob), 1/(2*prob)] ver.
# @prob: probability (access frequency)
# @x   : time
```

```
fregion := prob -> x -> piecewise(-1/prob/2 < x and x < 1/prob/2, prob, 0);
#------------------------------------------------------------------------------#

#------------ begin: CDF of processors ----------------------#
# integral of ditribution function (actually, just area calc. of rectangle)
# ver. region [-1/prob, 0]
# @prob: probability (access frequency)
# @st   : left edge of time region for integral
# @ed   : right edge of time region for integral
int_fregionpast_uni := proc(prob, st, ed) local stadj, edadj,tau;
    stadj := piecewise(
        st < -1/prob, -1/prob,
        st);
    edadj := piecewise(
        0 < ed, 0,
        ed);
    return piecewise(
        edadj < stadj, 0,
        0 < stadj, 0,
        edadj < -1/prob, 0,
        fregionpast(prob)(0) * (edadj - stadj)
        );
end proc;

# ver. region [-1/(2*prob), 1/(2*prob)]
# @prob: probability (access frequency)
# @st   : left edge of time region for integral
# @ed   : right edge of time region for integral
int_fregion_uni := proc(prob, st, ed) local stadj, edadj,tau;
    stadj := piecewise(
        st < -1/prob/2, -1/prob/2,
        st);
    edadj := piecewise(
        1/prob/2 < ed, 1/prob/2,
        ed);
    return piecewise(
        edadj < stadj, 0,
        1/prob/2 < stadj, 0,
        edadj < -1/prob/2, 0,
```

```
        fregion(prob)(0) * (edadj - stadj)
        );
end proc;
#-------------------------------------------------------------------#
```

B.2 FCFS MODEL

The FCFS model $W_{FCFS}(f, n, k, t, d)$ is defined as `wFcfsK(prob, n, k, t, z)`. The companion function $I_{FCFS}(f, n, k, t)$ is defined as `iFcfs(prob, n, k, t)`. `wFcfsE(prob, n, t)` calculates expected values of `wFcfsK(prob, n, k, t, z)`.

```
# [[probability of k processes cross over the time t]]
# @prob: probability (access frequency)
# @n   : the number of processors
# @k   : the number of accesses which interfere the access in focus
# @t   : the time when the processor in focus access the memory
iFcfs := proc(prob, n, k, t) option cache;
    return piecewise(
        n = k and k > 0,  int_fregionpast_uni(prob, t-1, t)^ n,
        n = k and k = 0,  1,
        n > k and k = 0,  int_fregionpast_uni(prob, -infinity, t)^n
                          - add( thisproc(prob,n,i,t), i=1..n),
        n > k and k > 0, thisproc(prob,n,k+1,t-1)
                         + add(
                            binomial(n,i) * thisproc(prob,i,i,t)
                            * thisproc(prob,n-i, k-i+1,t-1)
                            , i=1..k)
                         + binomial(n,k) * thisproc(prob,k,k,t)
                         * thisproc(prob,n-k, 0, t-1)
    )
end proc;

# [[distribution function of delay time z]]
# @prob: probability (access frequency)
# @n   : the number of processors
# @k   : the number of accesses which interfere the access in focus
# @t   : the time when the processor in focus access the memory
```

```
# @z   : duration which the processor in focus is interfered
wFcfsK := proc(prob, n, k, t, z) option cache;
    local x;
    x := z-k+1;
    return piecewise(
        z = 0 and k = 0, iFcfs(prob,n, 0, t),
        z < 0 or (k = 0 and z != 0) or (k != 0 and (x <= 0 or 1 < x)), 0,
        k-1 < z and z <= k, piecewise(
            n = k,
                n * fregionpast(prob)(t-1+x) * int_fregionpast_uni(prob,t-1+x,t)^(n-1),
            n > k,
                thisproc(prob, n, k+1, t-1, (k+1)-1+x)
                + add(
                    binomial(n, i) * thisproc(prob, n-i, k-i+1, t-1,(k-i+1)-1+x)
                    * int_fregionpast_uni(prob,t-1,t)^i
                    , i=1..k-1)
                + binomial(n, k) * thisproc(prob, n-k, 1, t-1, x)
                * add(
                    binomial(k, j) * int_fregionpast_uni(prob,t-1,t-1+x)^j
                    * int_fregionpast_uni(prob,t-1+x,t)^(k-j)
                    , j=1..k)
                + binomial(n, k) * (int(thisproc(prob, n-k, 1, t-1,tau), tau=0..x)
                + iFcfs(prob,n-k, 0, t-1)) * k * fregionpast(prob)(t-1+x)
                * int_fregionpast_uni(prob,t-1+x,t)^(k-1)
              )
        );
end proc;

# [[function to obtain expected values]]
# @prob: probability (access frequency)
# @n   : the number of processors
# @t   : the time when the processor in focus access the memory
wFcfsE := (prob, n, t) -> add(int(zt * wFcfsK(prob, n, k, t, zt), zt=k-1..k), k=1..n);
```

B.3 FP MODEL

The FP model $W_{FP}(f, p, n, k, t, d)$ is defined as `wcPriK(prob, p, n, k, t, z)`. The FP model for the lowest priority processor $W_{FP_l}(f, n, k, t, d)$ is defined as `wcLowestK(prob, n, k, t, z)`. `wcPriE(prob, p, n, t)` calculates expected values of `wcPriK(prob, p, n, k, t, z)`. Other functions are companion functions for above functions.

```
# [preparation 1]
# @prob: probability (access frequency)
# @n   : the number of processors
# @k   : the number of accesses which interfere the access in focus
# @t   : the time when the processor in focus access the memory
ipast := proc(prob, n, k, t) option cache;
    return piecewise(
        n = k and k > 0,  int_fregion_uni(prob, t-1, t)^ n,
        n = k and k = 0,  1,
        n > k and k = 0,  int_fregion_uni(prob, -infinity, t)^n
                          - add( thisproc(prob,n,i,t), i=1..n),
        n > k and k > 0, thisproc(prob,n,k+1,t-1)
                         + add(
                            binomial(n,i) * thisproc(prob,i,i,t)
                            * thisproc(prob,n-i, k-i+1,t-1)
                            , i=1..k)
                         + binomial(n,k)
                         * thisproc(prob,k,k,t) * thisproc(prob,n-k, 0, t-1)
    )
end proc;

# [[probability of k processes cross over the time t]]
# [note] use fregion instead of fregionpast
# @prob: probability (access frequency)
# @n   : the number of processors
# @k   : the number of accesses which interfere the access in focus
# @t   : the time when the processor in focus access the memory
ipast_complete := proc(prob, n, k, t) option cache;
    return ipast(prob,n,k,t)
```

```
          + add(
               binomial(n, i) * ipast(prob,i, k, t)
               * int_fregion_uni(prob,t,infinity)^(n-i)
               , i=0..n-1);
end proc;

# [[distribution function of delay time z]]
# [note] use fregion instead of fregionpast
# @prob: probability (access frequency)
# @n   : the number of processors
# @k   : the number of accesses which interfere the access in focus
# @t   : the time when the processor in focus access the memory
# @z   : duration which the processor in focus is interfered
wpastK := proc(prob, n, k, t, z) option cache;
    local x;
    x := z-k+1;
    return piecewise(
        z = 0 and k = 0, ipast(prob,n, 0, t),
        z < 0 or (k = 0 and z != 0) or (k != 0 and (x <= 0 or 1 < x)), 0,
        k-1 < z and z <= k, piecewise(
            n = k, n * fregion(prob)(t-1+x) * int_fregion_uni(prob,t-1+x,t)^(n-1),
            n > k,
                thisproc(prob, n, k+1, t-1, (k+1)-1+x)
                + add(
                    binomial(n, i) * thisproc(prob, n-i, k-i+1, t-1,(k-i+1)-1+x)
                    * int_fregion_uni(prob,t-1,t)^i
                    , i=1..k-1)
                + binomial(n, k) * thisproc(prob, n-k, 1, t-1, x)
                * add(
                    binomial(k, j) * int_fregion_uni(prob,t-1,t-1+x)^j
                    * int_fregion_uni(prob,t-1+x,t)^(k-j)
                    , j=1..k)
                + binomial(n, k) * (int(thisproc(prob, n-k, 1, t-1,tau), tau=0..x)
                + ipast(prob,n-k, 0, t-1)) * k
                * fregion(prob)(t-1+x) * int_fregion_uni(prob,t-1+x,t)^(k-1)
              )
        );
end proc;
```

```
# [preparation 2]
# probability of k access by higher priority PEs in [tpre, t] cross over the time t
# @prob: probability (access frequency)
# @n   : the number of processors
# @k   : the number of accesses which interfere the access in focus
# @tpre: the left edge of time region
         when at least one remaining processor should access the memory
# @t   : the time when the processor in focus access the memory
icLowestInner := proc(prob, n, k, tpre, t) option cache;
    return piecewise(
        tpre >= t, 0,
        piecewise(
            n = k and k = 0 , 1,
            n < k, 0,
            k = 0, int_fregion_uni(prob, t, infinity)^n,
            k >= 1, add(
                        binomial(n,i) * int_fregion_uni(prob,tpre,t)^i
                        * thisproc(prob, n-i,k-i,t, t+i)
                        , i=1..k)
        )
    );
end proc;

# [[probability of k access interfere the access by the PE with the lowest priority]]
# @prob: probability (access frequency)
# @n   : the number of processors
# @k   : the number of accesses which interfere the access in focus
# @t   : the time when the processor in focus access the memory
icLowest := proc(prob, n, k, t) option cache;
    return piecewise(
        k = 0,
            ipast(prob,n,0,t)
            + add(
                binomial(n, i) * ipast(prob,i, 0, t)
                * int_fregion_uni(prob,t,infinity)^(n-i)
                , i=0..n-1),
        k >= 1,
            ipast(prob,n,k,t)
            + add(add(
```

```
                binomial(n,i) * ipast(prob,i, l, t)
                * int(icLowestInner(prob,n-i, k-l, t, t+l-1+rest), rest=0..1)
                , i=l..n-1), l=1..k)
    );
end proc;

# [[distribution function of delay time z for the PE with the lowest priority]]
# @prob: probability (access frequency)
# @n   : the number of processors
# @k   : the number of accesses which interfere the access in focus
# @t   : the time when the processor in focus access the memory
# @z   : wait time
wcLowestK := proc(prob, n, k, t, z) local x;
    x := z-k+1;
    return piecewise(
        z = 0 and k = 0, icLowest(prob,n,0,t),
        z = 0 and k > 0, 0,
        k >= 1, wpastK(prob,n,k,t,k-1+x)
                + add(add(
                    binomial(n,i) * wpastK(prob, i, l, t,l-1+x)
                    * icLowestInner(prob, n-i, k-l, t, t+l-1+x)
                    , i=l..n-1), l=1..k)
        );
end proc;

# [[definition of permutation nPk]]
Pnk := (n, k) -> binomial(n, k)*k!;

# [[FP model main: distribution function of delay time z]]
# @prob: probability (access frequency)
# @p : priority of the processor in focus (0 <= p < n)
# @n   : the number of processors
# @k   : the number of accesses which interfere the access in focus
# @t   : the time when the processor in focus access the memory
# @z   : duration which the processor in focus is interfered
wcPriK := proc (prob, p, n, k, t, z) local x; option cache;
    x := z-k+1;
    return piecewise(
        p = 0, piecewise(
```

```
    k = 0, wcLowestK(prob, n, 0, t, 0),
    k = 1, add(wcLowestK(prob, n, i, t, i-1+x), i = 1 .. n),
    2 <= k, 0),
1 <= p and p < n, piecewise(
    k = 0, wcLowestK(prob, n, 0, t, 0),
    1 <= k and k < p+1,
        add(
            add(
            wpastK(prob, j, i, t, i-1+x)
            *(
                binomial(n-p, j)
                *icLowestInner(prob, p, k-1, t, t+x)
                *int_fregion_uni(prob, t, infinity)^(n-p-j)
                +
                add(
                    binomial(p, h)*binomial(n-p, j-h)
                    *add(piecewise(
                        j-h >= i-m,
                            binomial(h, m)*binomial(j-h, i-m)/binomial(j, i)
                            * piecewise(
                            m > k,
                                0,
                            m < i and m < k,
                                (m/i)
                                *icLowestInner(prob, p-h, k-m, t, t+m-1+x)
                                +
                                ((i-m)/i)
                                * icLowestInner(prob, p-h, k-m-1, t, t+m+x),
                            m < i and m = k,
                                (m / i)
                                *int_fregion_uni(prob, t+k-1+x, infinity)^(p-h),
                            m = i and m <= k,
                                icLowestInner(prob, p-h, k-i, t, t+i-1+x)
                            ),
                        0)
                    , m = 0 .. min(h, i))
                    * int_fregion_uni(prob, t, infinity)^((n-p)-(j-h))
                , h = 1 .. min(p, j))
            ), i = 1 .. j), j = 1 .. n),
```

```
            k = p+1,
                add(add(
                    wpastK(prob, j, i, t, i-1+x)
                    *add(piecewise(
                        h < i,
                            binomial(p, h)*binomial(n-p, j-h)
                            *binomial(h, h)*binomial(j-h, i-h)/binomial(j, i)
                            *(i-h) / i
                            *icLowestInner(prob, p-h, p-h, t, t+(h+1)-1+x)
                            *int_fregion_uni(prob, t, infinity)^(n-p-j+h),
                        i <= h, 0)
                     , h = 0 .. min(p, j-1))
                , i = 1 .. j), j = 1 .. n),
            p+1 < k, 0
            ),
        p = n, wcLowestK(prob, n, k, t, k-1+x))
end proc;

# [[function to obtain an expected value]]
# @prob: probability (access frequency)
# @p : priority of the processor in focus (0 <= p < n)
# @n   : the number of processors
# @t   : the time when the processor in focus access the memory
wcPriE := (prob, p, n, t) -> add(int(z * wcPriK(prob,p,n,k,t,z), z=k-1..k), k=1..n);
```

B.4 RR MODEL

RR model $W_{RR}(f, n, k, t, d)$ is defined as `wcRRK(prob, n, k, t, z)`.

`wcRRE(prob, n, t)` calculates expected values using `wcRRK(prob, n, k, t, z)`.

```
# [[distribution function of delay time z]]
# @prob: probability (access frequency)
# @n   : the number of processors
# @k   : the number of accesses which interfere the access in focus
# @t   : the time when the processor in focus access the memory
# @z   : duration which the processor in focus is interfered
wcRRK := proc(prob, n, k, t, z)
```

```
    return piecewise(
        k = 0, wcLowestK(prob, n, 0, t, 0),
        k > 0, add(iFcfs(prob, n, pf, t) * wcPriK(prob, pf, n, k, t, z), pf=0..n)
    );
end proc;

# [[function to obtain an expected value]]
# @prob: probability (access frequency)
# @n   : the number of processors
# @t   : the time when the processor in focus access the memory
wcRRE := (prob, n, t) -> add(int(z * wcRRK(prob,n,k,t,z), z=k-1..k), k=1..n);
```