Automatic Instrumentation of Profilers for FPGA-based Design Space Exploration

Seiya Shibata [#*1], Yuki Ando [#2], Shinya Honda [#3], Hiroyuki Tomiyama [#4], Hiroaki Takada [#5]

[#] *Graduate School of Information Science, Nagoya University, Nagoya 464-8603, Japan*

{[1]shibata, [2]y_ando, [3]honda, [4]tomiyama, [5]hiro}@ertl.jp

[*] *The Japan Society for the Promotion of Science*

***Abstract*—In the system-level design of MPSoCs (Multi-Processor System-on-a-Chips), system designers start from describing functionalities of the system as processes and channels, and then decide mapping of them to various Processing Elements (PEs) including CPUs and dedicated hardware modules. A mapping decision is evaluated by simulation or FPGA-based prototyping. Designers iterate mapping and evaluation until all design requirements are met. 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 channel accesses. According to mapping of processes to PEs, the profilers are automatically configured and instrumented into FPGA-based system prototypes by a system-level design tool that we have developed. Designers therefore need to manually modify neither the system description nor the profilers upon each change of process mapping. In order to demonstrate the effectiveness of our profilers, a case study on MPEG4 decoder design was conducted.**

I. 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. In the system-level design, system designers start from describing functionalities of the system as processes and channels which indicate computations and communications among processes, respectively. Then the designers decide mapping of them to various Processing Elements (PEs) including CPUs and dedicated hardware modules [1]. A mapping decision is evaluated by simulation or FPGA-based prototyping. The designers iterate mapping and evaluation until all design requirements are met.

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 [2][3][4]. These tools convert processes and channels into compilable software programs and synthesizable RTL circuits depending on their mapping. The generated timed descriptions can be evaluated by simulation or FPGA-based prototyping.

A number of researches on simulation-based evaluation were conducted in the past [5][6][7] and cycle-accurate hardware simulation tools [8][9] were widely accepted in the industrial domain. However, since there are a trade-off between speed and accuracy on simulations on a host PC, simulations are often inappropriate for design space exploration which needs both speed and accuracy, 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 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 channel accesses. In our framework, systems are described at a high level and FPGA-based system prototypes are automatically synthesized by our system-level design tool, named SystemBuilder. According to mapping of processes to PEs, the profilers are automatically configured and instrumented into the FPGA-based system prototypes by SystemBuilder. 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 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 multi-processor systems (MPSoCs).

The rest of this paper is organized as follows. First, section II explains our system-level design toolkit and then section III describes two proposed profilers. Section IV shows the effectiveness of the profilers through a case study. Finally Section V concludes this paper with a summary.

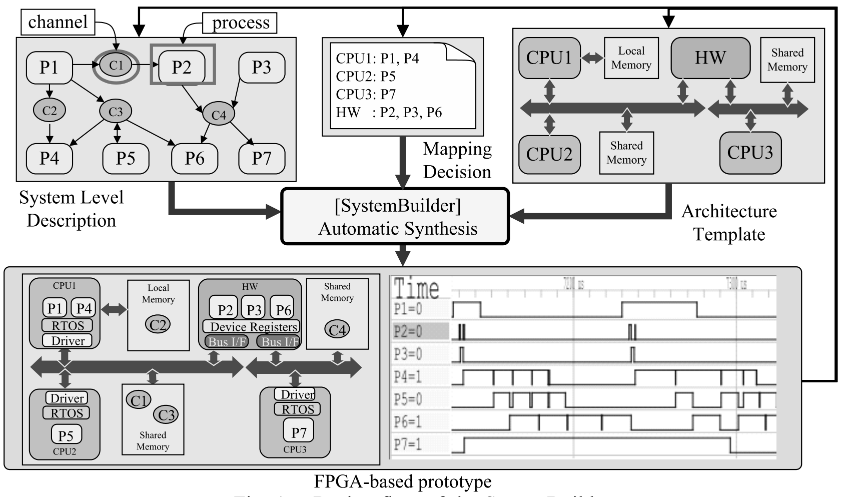


Fig. 1. Design flow of the SystemBuilder

II. SystemBuilder

SystemBuilder is a system-level design toolkit developed in our prior work [3]. In this work, we extended SystemBuilder to automatically instrument the systems with a process profiler and a memory profiler. In this section, we explain a brief overview of the design flow achieved by SystemBuilder.

Fig. 1 shows the design flow with SystemBuilder. First, a system designer develops a “system level description” to capture functionalities of the target system. The system level description consists of “processes” and “channels”. Processes are described in the C language and represent computation components of the system. Channels are provided as C-APIs by SystemBuilder and represent inter-process communications at a high abstraction level. Currently SystemBuilder provides three types of channels: blocking channels, register channels and memory channels. Blocking channels are used for synchronization between two processes, and register channels and memory channels are used for storage of data which are sent/received among processes.

The processes may be mapped onto CPUs and hardware modules, and the channels onto buses, memories and other communication devices. The designer also specifies mapping of the processes and the channels. SystemBuilder automatically synthesizes descriptions of interconnections between processes. The synthesized communication descriptions are generated in the C language and VHDL, depending on mapping of the processes and the channels. Next, SystemBuilder makes use of a cross-compiler of the CPUs for software and a behavioral synthesis tool for hardware module in order to obtain an executable binary and synthesizable RTL circuits, respectively. Finally, a configuration bitstream of hardware architecture for an FPGA is synthesized by a logic synthesis tool from the RTL circuits and IPs of the CPUs and essential peripherals.

III. System-Level Profilers

We propose two profilers, a process profiler and a memory profiler, which are automatically configured and instrumented into FPGA-based system prototypes by SystemBuilder. In this section, we describe detail of the profilers.

A. Profiling Flow

The introduction of the profilers to the design flow of SystemBuilder is easy. After a system designer describes the functionalities in “system level description” (shown in Fig. 1), SystemBuilder automatically generates an FPGA-based prototype. If the designer turns on the profiler synthesis option of SystemBuilder at the prototype generation, the process profiler and the memory profiler are automatically configured and instrumented in the prototype. Process trace and memory trace are recorded on execution of the prototype. Finally, the traces are transferred from the FPGA to the host PC and the designer analyzes the system using them.

Since capacities of memories are limited, the profilers cannot record traces of an entire system execution. SystemBuilder therefore 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.

B. The Process Profiler

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

Fig. 2 illustrates the overall structure of our profilers. The process profiler consists of processes which are instrumented for profiling and “process profiler module” hardware (denoted on the upper right side of “hardware module” in Fig. 2). The process profiler module consists of “process trace extractor”, “process trace writer”, a timer module, a FIFO and a memory module. At a runtime of the system, processes send signals to registers of the trace extractor. The process trace extractor periodically collects the values of the registers, and sends them to the process trace writer through the FIFO. The process trace writer writes data to the memory module whenever it receives values from the FIFO. Since we made a dedicated memory module and a dedicated access interface for the memory

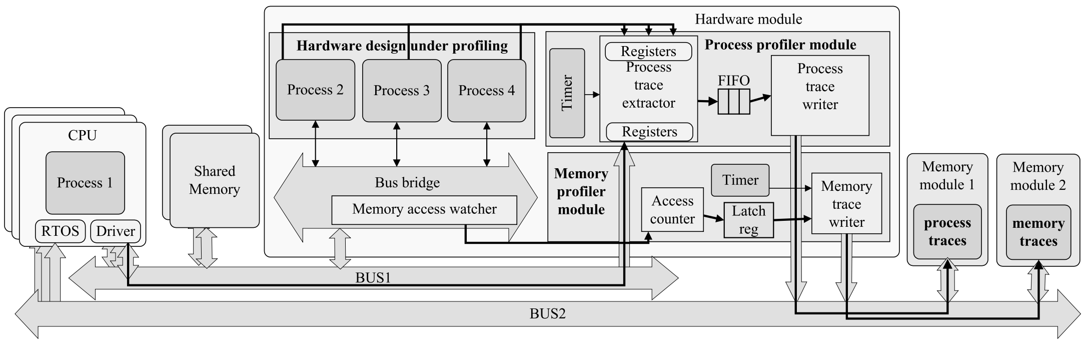


Fig. 2. Overall structures of the process profiler and the memory profiler

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. The transformed channel accesses write “0” signal to registers of the trace extractor at the beginning of their blocking communication, and write “1” signal at the completion of the communication.

After the execution, the trace data are read from the memory module and are output to a host PC. The process profiler also provides the trace analyzer for the traces obtained from an FPGA. The trace analyzer generates a VCD (Value Change Dump) file. The VCD file can be visualized as waveforms using tools such as *GTKWave* [11]. In the waveforms, high states mean execution 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.

C. 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, which is illustrated on the lower right side of the hardware module in Fig. 2. We designed the memory profiler focusing the feature that all processes mapped on hardware accesses outside memories through “bus bridges” synthesized by SystemBuilder (illustrated in Fig. 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. For each period, the memory trace writer sends the sums of access/blocked cycles to the dedicated memory module.

After profiling, the traces are read from the memory module and are transferred to a host PC. The trace analyzer generates various graphs which show the entire trace and periodic changes.

IV. A Case Study: MPEG4 Decoder System Design

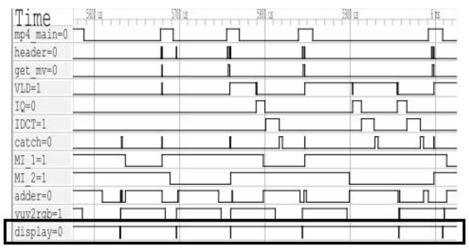
In order to demonstrate the effectiveness of a process profiler and a memory profiler, we show a case study of MPEG4 decoder system design. The case study was performed targeting an Altera Stratix II FPGA board with Nios II soft-core CPUs at 50 MHz of clock frequency. eXCite 3.2a [12] was used for behavioral synthesis, and logic synthesis and P&R were done by Quartus 8.1.

We designed an MPEG4 decoder system with SystemBuilder. Based on the MPEG4 decoder developed in the past [13], we improved performance of the design using the process profiler and the memory profiler.

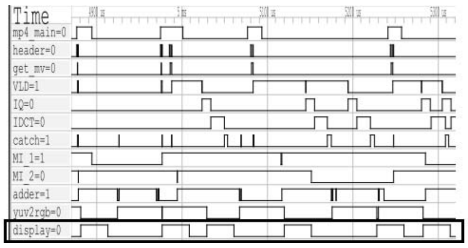
The MPEG4 decoder consists of 11 processes, *mp4_main*, *header*, *get_mv*, *VLD*, *IQ*, *IDCT*, *MI-1*, *MI-2*, *adder*, *yuv2rgb* and *display*. The mp4_main process handles inputs and the display process outputs decoded images to VRAM of a VGA device. The mp4_main process should be implemented in 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 further improvements. We therefore used the process profiler in order to find the bottleneck processes. Fig. 3(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. 4 illustrates a graph obtained by the memory profiler. White bars on the back side of Fig. 4 illustrates the sums



(a) MPEG4 decoder with a single bus interface



(b) MPEG4 decoder with two bus interfaces

Fig. 3. The process profiler results of two MPEG4 decoder implementations

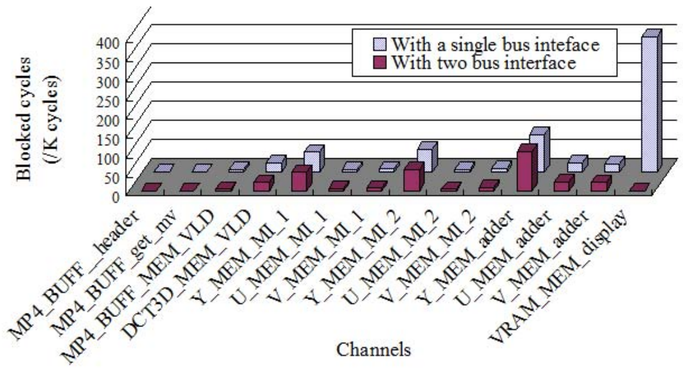


Fig. 4. Sums of blocked cycles for each channels

of blocked cycles (on x-axis) for individual channels which access off-chip memories (on y-axis) of the all hardware implementation. From the rightmost bar of Fig. 4, we could see that the “VRAM_MEM_display” channel which transfers decoded images to VRAM of the VGA device was frequently blocked by other channels. This indicates 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 are three points at which conflicts are 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. 2) which manages bus accesses from processes mapped 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 bridge. We solved the conflicts by making a special bus bridge for the display process. 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 bridge is made. As a result, the special bus bridge enabled the display process to access the VRAM exclusively. We obtained the process profile shown in Fig. 3(b) and memory profile shown as black bars in front side of Fig. 4. In comparison with waveforms at the bottom of Fig. 3(a), that of Fig. 3(b) shows that the execution time (high state time in the waveform) of the display process was reduced. The memory profile shown in Fig. 4 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.

V. Conclusions

We proposed two profilers, a process profiler and a memory profiler, for design space exploration at a system-level. The process profiler records activation/wait timings of processes and the memory profiler records access/blocked cycles of shared memory accesses. The profilers are automatically instrumented into the FPGA-based system prototypes by our system-level design toolkit. Automatic instrumentation of the profilers enables smooth iterations of evaluation.

We demonstrated the effectiveness of the profilers through a case study on MPEG4 decoder system design. The case study presented a performance improvement example using the profilers considering parallelism of processes and conflicts at memory accesses.

Acknowledgments

This work is in part supported by STARC (Semiconductor Technology Academic Research Center).

References

[1] K. Keutzer, S. Malik, A. R. Newton, J. M. Rabaey, and A. Sangiovanni-Vincentelli, “System level design: orthogonalization of concerns and platform-based design,” *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 12, 2000.

[2] 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, no. 3, 2008.

[3] S. Honda, H. Tomiyama, and H. Takada, “RTOS and codesign toolkit for multiprocessor systems-on-chip,” *ASP-DAC*, 2007.

[4] 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,” *CODES+ISSS*, 2007.

[5] S. Honda, T. Wakabayashi, H. Tomiyama, and H. Takada, “Rtos-centric cosimulator for embedded system design,” *IEICE Trans. Fundamentals*, vol. E87-A, no. 12, 2004.

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

[7] 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, 2009.

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

[9] CoWare Inc. http://www.coware.com/.

[10] 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. Design Automation of Electronic Systems*, vol. 12, no. 3, 2007.

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

[12] Y Explorations, Inc. http://www.yxi.com/index.html.

[13] S. Shibata, S. Honda, H. Tomiyama, and H. Takada, “A case study of MPEG4 decoder design with SystemBuilder,” *VLSI-DAT*, 2009.