A Case Study on MPEG4 Decoder Design with SystemBuilder

Seiya Shibata, Shinya Honda, Hiroyuki Tomiyama and Hiroaki Takada
Graduate School of Information Science
Nagoya University
Nagoya 464-8603, Japan
{shibata, honda, tomiyama, hiro}@ertl.jp

Abstract— This paper presents a case study on designing an MPEG4 decoder system using our system-level design toolkit named SystemBuilder. We start with a sequential specification of the MPEG4 decoder behavior and generate an FPGA implementation. In order to improve the performance, we refine the behavioral description based on the analysis result obtained by a profiler. Finally, we achieve over 15fps performance with pipelined hardware implementation.

I. Introduction

System-on-a-Chip (SoC) is a key technology for the embedded domains. Advances on manufacturing technologies have enabled more hardware components to be integrated into one chip, so that larger scale applications can be executed on it. Due to the advances and increasing demands for large applications, embedded systems to be developed have never been more complex. Therefore needs for tools which lift up design capabilities have been increasing.

Recent growth of high-level synthesis (HLS) tools has enabled hardware designers to develop hardware modules at behavioral level using C/C++ like languages [1]. By means of compilers and HLS tools, whole system can be described in a single behavioral language in short time. However, explorations of system architecture should be done properly at system-level which is higher than software/hardware-level, to obtain optimized software/hardware partitioning and communications between them.

Various researches have been conducted on system-level design tools. PeaCE [2] is a system design tool featuring implementation generation from models. System modeling should be done by designers using modeling method specially designed for PeaCE. ARTS [3] provides a simulation platform for multi-processor SoCs modeled in SystemC. It supports multiple processing element (PE) models and network model among PEs. ARTS assumes that the application model simulated on it is already developed and separated properly in order to explore allocation to PEs. In industrial system design processes, however, systems are often developed from existing sequential programs by converting them. Therefore embedded system design should consider system development from sequential programs.

In prior work, we developed a system-level design toolkit, named *SystemBuilder* [4]. The main objective of SystemBuilder is to help designers explore software/hardware partitioning efficiently, based on iterative evaluations by executing systems on a target FPGA. SystemBuilder takes system-level description written in the C language and mapping specification to architectures as input, and automatically generates target implementations including software, hardware and interfaces among them. Since communication interfaces should be developed on every change of software/hardware partitioning decision, the interface synthesis capability especially affects design time. SystemBuilder automatically generates interface implementations and enables designers to avoid describing them such as hardware driver programs and hardware control logics. With this capability, system designers need not consider details of interfaces and can develop system-level description easily.

In this paper, we show a case study on designing an MPEG4 decoder system with using SystemBuilder. MPEG4 decoder is an industry-strength application used in video cameras and cellphones, 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 MPEG4 decoder system achieving 15fps (frames per second) performance, and show effectiveness of SystemBuilder on system-level design.

This paper is organized as follows. Section II explains a brief overview of SystemBuilder. Sections III and IV present a case study on MPEG4 decoder system design. Section V evaluates effectiveness and problems of SystemBuilder clarified through our case study, and Section VI concludes this paper.

II. SystemBuilder

In this section, we show a brief overview of SystemBuilder. Please refer to [4] for the detail of SystemBuilder.

A. Input description

Figure 1 shows the mapping and synthesis overview 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, 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. Communication APIs used in each process description are converted to interface programs/logics to communicate with each other through channels.

SystemBuilder synthesizes target implementations automatically by mapping system described in SLD to the architecture template. Note that mapping decision should be done by a system designer.

B. Verification and Analysis

SystemBuilder generates implementations of a system for not only a target FPGA but also a cosimulation platform. We developed a software/hardware cosimulation platform in prior works [5]. Our cosimulation platform enables a designer to verify functionalities of systems that consist of descriptions at multiple abstraction levels.

978-1-4244-2782-6/09/$25.00 ©2009 IEEE

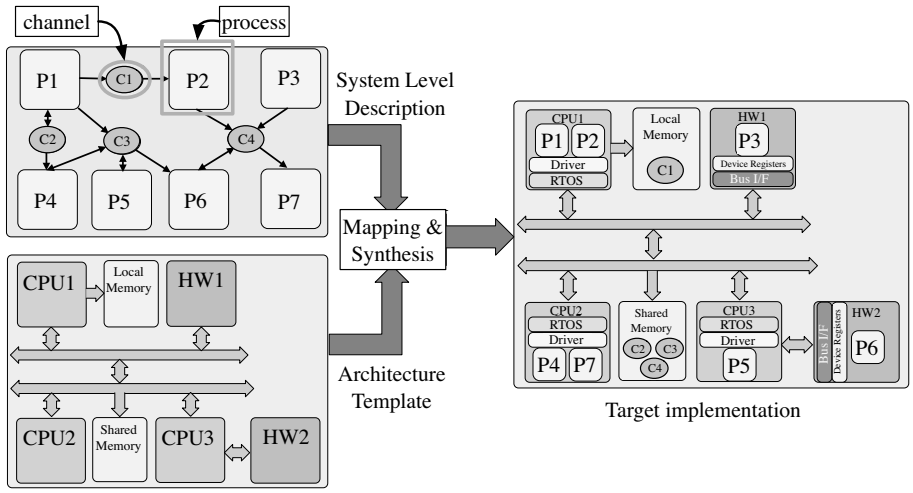


Figure 1. Mapping and synthesis by SystemBuilder.

Figure 2. Example of process profiler waveform.

In order to refine system within short time, SystemBuilder provides a set of analysis tools (denoted as "process profiler" hereafter). Process profiler helps a designer find performance bottlenecks out from processes executing concurrently by visualizing execution histories in waveforms. Execution histories consists of activation/suspension timings of each process. Figure 2 shows an example of waveform available with process profiler. Process profiler gathers histories of both software processes and hardware processes on a target FPGA, and shows them on a PC. Using process profiler, designers can easily decide software/hardware partitioning and find out processes to be optimized.

III. MPEG4 Decoder SLD

Our case study aims to make an MPEG4 decoder system achieve 15fps performance on a target FPGA.

This section shows the efficiency on developing SLD through our case study. We start MPEG4 decoder system design from modifying a sequential software program into SLD. The software program of MPEG4 decoder is selected from EEMBC benchmark suite [6]. At the end of this section, we obtain SLD where most processes can be implemented as hardware and executed concurrently in pipeline manner.

A. 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 TOPPERS/JSP kernel, which is a popular RTOS in Japanese industries. Processes specified as hardware are converted to RTL (Register Transfer Level) description by an HLS tool, as which we used a commercial tool, YXI eXCite 3.2a[7]. 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 generate maximum clock frequencies available on designed systems.

B. Initial Decision

We first decided the specifications of MPEG4 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, MPEG4 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.

C. SLD Construction

The minimum SLD is constructed of a single process and no channel (illustrated in Figure 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 MPEG4 decoder system with a single process that decodes MPEG4 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 3(b) illustrates the MPEG4 decoder system after separating one process from top process. We first used GNU profiler (*gprof*) for analyzing bottlenecks since initial system consists of a single process and the process profiler cannot analyze the system. 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

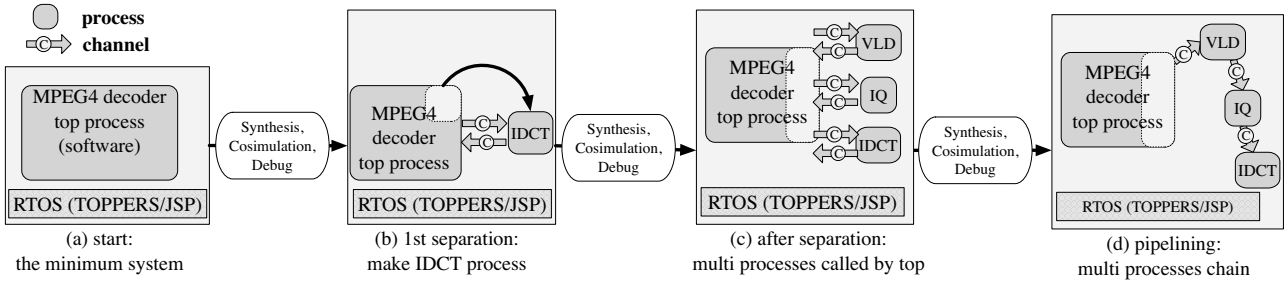


Figure 3. Incremental process separation.

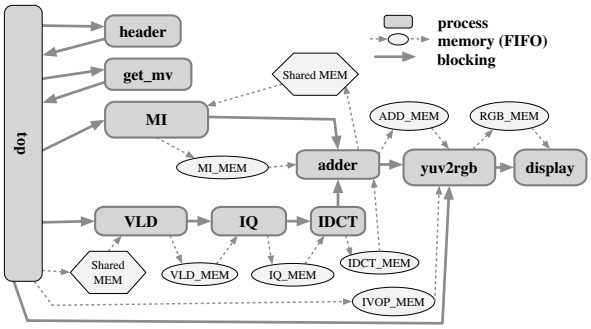


Figure 4. Pipelined system structure for coded-P-VOP.

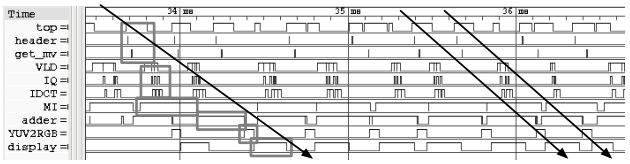


Figure 5. Wave form of pipelined decoder.

Figure 6. Profile of P-VOP decoding.

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 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 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 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-VOP, the system has channels between top process and yuv2rgb process (illustrated as I_VOP_MEM and a solid arrow in Figure 4). I- and not-coded-P-VOP are decoded by top process, and transferred to yuv2rgb process through the channels.

Table I shows decoding time, frame rates and performance improvements on each refinement. Required ALUTs (adaptive look-up tables) and MEMs (block memories) used in Stratix II FPGA are also shown in the table. "Software implementation" where all processes are implemented as software decoded input files at no more than 3 fps performance. "Hardware implementation" where most processes are implemented as hardware forms pipelined hardware controlled by top process, and achieves approximately 5× performance compared with software implementation. While clock frequency was reduced to 75MHz on hardware implementation due to the complexities of hardware processes, it achieved over 13 fps for "marsface" and 6 fps for "railgrind". So far, pipeline structured system description was developed within 2 weeks.

IV. Refinement for Performance

In order to achieve 15fps performance on the MPEG4 decoder system, we analyzed pipeline behavior of processes and refined SLD.

A. Analysis of Process Behavior

Analyses are performed with using the process profiler. Figure 5 shows a snapshot of waveforms representing processes behaviors. In the waveform, processes are listed in control flow order. We can see that each process forms pipeline execution as we expected, which is annotated with arrows in Figure 5. However, looking at entire waveform illustrated in Figure 6, we found that not-coded-P-VOPs should also be accelerated the same as coded-P-VOPs. For this reason, we decided to modify the hardware processes and enable them to handle both coded-P-VOPs and not-coded-P-VOPs.

B. VOPs Merging

Not-coded-P-VOP was decoded on top process in the SLD developed in Section III. Although actual work of decoding not-coded-P-VOP is only to copy image blocks, memory address calculation is expensive for software processes and it was expected to accelerate execution by hardware implementation. Because of the similarity of their programs between not-coded- and coded-P-VOPs, we could easily extract programs for not-coded-P-VOPs from top process and merge them with existing hardware processes. Moreover we also merged I-VOP decoding to existing hardware processes because of the similarity.

Figure 7 illustrates overview of waveform for decoding a frame. We can see that two more hardware processes (MI and adder) are used for decoding not-coded-P-VOPs. This improvement (denoted as "+VOPs merging" in Table I) achieved approximately 6× performance compared with software implementation. Additionally, VOPs merging also led to reduce the number of required ALUTs and memories on the target FPGA, since two channels between top and yuv2rgb processes became unnecessary and were removed due to the merging described above. Available clock frequency was also raised to 81.25MHz due to the logic simplification. So far, system performance achieved over 15fps for "marsface" but not for "raildrind". This transformation was performed in a day including synthesis and verification.

C. Memory Access Reduction

Figure 8(a) illustrates a part of Figure 7. We found out that two processes of MI and adder are always active and keep other processes

TABLE I. PERFORMANCE IMPROVEMENT OF AN MPEG4 DECODER SYSTEM ON ITERATIVE REFINEMENTS.

Design	clock (MHz)	exec. time (sec)		frame rate (fps)		improvement		hardware usage	
		marsface	railgrind	marsface	railgrind	marsface	railgrind	ALUT	MEM
software implementation	100.00	16.4	70.5	3.0	1.4	1.0x	1.0x	5,427	1,112,576
hardware implementation	75.00	3.6	15.9	13.6	6.1	4.5x	4.4x	31,748	1,686,792
+VOPs merging	81.25	2.8	10.8	17.3	9.0	5.8x	6.5x	32,128	1,637,640
+memory access reduction	90.00	1.4	6.0	34.6	16.1	11.6x	11.7x	33,964	1,638,080

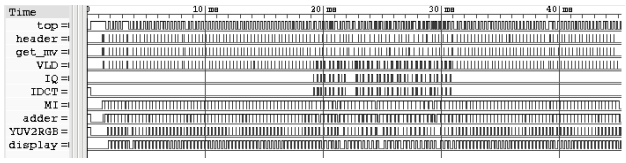
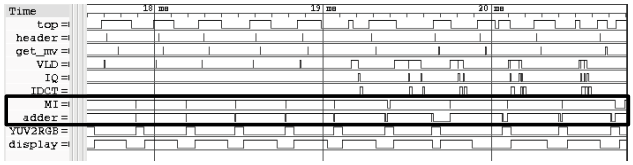
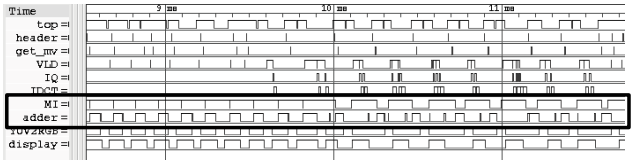


Figure 7. Profile after VOPs merging.



(a) Profile before.



(b) Profile after.

Figure 8. Effects of memory access reduction.

waiting. Obviously the two processes were bottlenecks that prevent from achieving 15fps performance. At this step, in order to improve performance of these processes, we analyzed in detail and refined their programs.

In the MPEG4 decoder, input images consists of pixels represented in 8 bits. In original software programs, pixels were loaded from and stored to memories one by one in 8 bits. In contrast, target architecture employs 32 bits width bus and memory interfaces. Because each load/store access leads process to wait long response time of bus and memories, we transformed the program manually to pack four 8-bit data into 32 bits for memory accesses. This transformation resulted in reducing the number of accesses to approximately one fourth.

After memory access reduction, we obtained waveform shown in Figure 8(b). We can see that execution period of two bottleneck processes are shortened roughly in half. This optimization took approximately a day. As a result, our MPEG4 decoder achieved over 15fps performance on 90MHz for 320×240 movies (shown as "+memory access reduction" in Table I).

V. SYSTEMBUILDER EVALUATION

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

We found five advantages through this case study as follows. (1) We could reuse a software program of MPEG4 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. (5) We could check visually the effect of our transformations with using process profiler (demonstrated in Section IV).

Subjects was also found in this case study. One of them is insufficiency of process profiler. Although process profiler played important part for finding processes to be improved, it lacked capabilities to detect inner bottlenecks of those processes. Since memory access reduction mentioned in Section IV was particularly effective, we are currently working on visualizing memory accesses together with process profiler results.

VI. CONCLUSIONS

This paper presented a case study on an MPEG4 decoder system design with our system-level design toolkit named SystemBuilder. SystemBuilder can generate target implementations of the system given as system-level description and architecture mapping specification. The MPEG4 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 designed a system which achieves over 15fps performance for 320×240 movies by hardware implementation and pipelining on an FPGA running at 90MHz speed.

The overall design took 5 weeks with a designer. The much iteration was enabled with a short turnaround time of the steps, which is brought by an automatic synthesis capability of SystemBuilder. Therefore we conclude that system-level design with SystemBuilder is efficient.

ACKNOWLEDGMENTS

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

REFERENCES

[1] K. Wakabayashi, "CyberWorkBench: Integrated Design Environment Based on C-Based Behaviour Synthesis and Verification," *VLSI-DAT*, 2005.

[2] S. Ha, et al., "PeaCE: A Hardware-Software Codesign Environment for Multimedia Embedded Systems," *ACM Trans. Design Automation of Electronic Systems. vol.12, no.3*, 2007.

[3] S. Mahadevan, et al., "ARTS: A SystemC-based framework for multiprocessor Systems-on-Chip modelling," *Design Automation for Embedded Systems, vol.11, no.4*, 2007.

[4] S. Honda, et al., "RTOS and Codesign Toolkit for Multiprocessor Systems-on-Chip, " *ASP-DAC*, 2007.

[5] S. Honda, et al., "RTOS-Centric Cosimulator for Embedded System Design," *IEICE Trans. Fundamentals, vol.E87-A, no. 12*, Dec. 2004.

[6] EEMBC, http://www.eembc.com/.

[7] Y Explorations, Inc., http://www.yxi.com/.