A Processor Core Synthesis System in IP-based SoC Design

Naoki TOMONO† Shunitsu KOHARA† Jumpei UCHIDA† Yuichiro MIYAOKA†
Nozomu TOGAWA‡,* Masao YANAGISAWA† Tatsuo OHTSUKI†
† Department of Computer Science, Waseda University
‡ Department of Information and Media Sciences, The University of Kitakyushu
* Advanced Research Institute for Science and Engineering, Waseda University.
email:tomono@yanagi.comm.waseda.ac.jp

Abstract— This paper proposes a new design methodology for SoCs reusing hardware IPs. In our approach, after system-level HW/SW partitioning, we use IPs for hardware parts, but synthesize a new processor core instead of reusing a processor core IP. System performs efficient parallel execution of hardware and software by taking account of a response time of hardware IP obtained by the proposed calculation algorithm. We can use optimal hardware IPs selected by the proposed hardware IPs selection algorithm. The experimental results show effectiveness of our new design methodology.

I. Introduction

The increased complexity of System-On-Chip (SoC) designs makes it difficult for designers to meet the demands from market such as short time-to-market, small gate count and high-performance. In practice, hardware/software co-design [1] and IP-based design [2] are proposed in order to build the required complexity in a short time. Another methodology is that after the hardware/software partitioning, a designer reuses the hardware IPs for hardware, and the processor core IPs for software.

However, the designers do not always find the suitable IPs that match the application. Some IPs have excess performance and some do not have enough performance.

In our approach, after the hardware/software partitioning, a designer uses hardware IPs for hardware, but synthesizes new processor core instead of reusing processor core IPs. Synthesizing processor core can compensate the excess or deficient performance of hardware IPs.

In this paper we propose a new processor core synthesis system which is hardware/software co-synthesis system based on response time of hardware IPs. A processor core synthesized by the system can execute another operation while hardware IPs execute the operations. The system can also select the suitable IP by a selection algorithm if there are some IPs which have the same functions but different performances.

Figure 1 shows a frame work based on the proposed processor core synthesis system. A designer describes the specification of application by systemC [3]. After evaluating and validating the performance required for the application, the designer decides which part of the specification is implemented by hardware or software (hardware/software partitioning). Then the hardware part is implemented by hardware IPs, and the software part is implemented by processor core synthesized by proposed system. The optimized processor core can absorb the excess or deficient performance of hardware IPs.

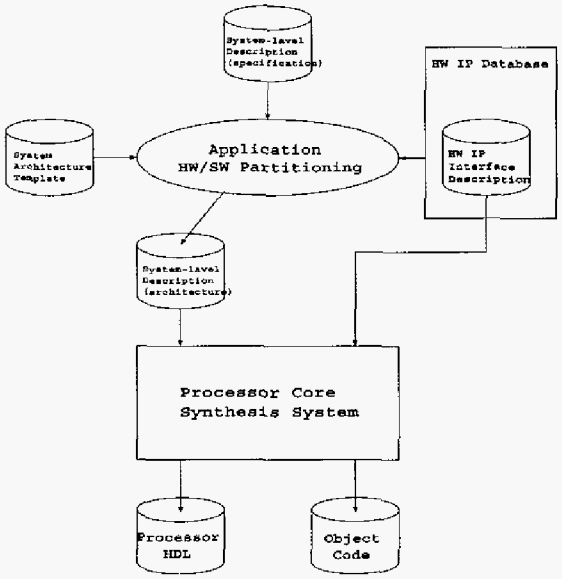

Fig. 1. A framework.

The system requires the response time of hardware IPs at the scheduling, but it was difficult to know the response time in advance. The designers selected one hardware IP by their experiences and intuitions if there were some hardware IPs which have the same functions but different performances.

In this paper, we propose a calculation algorithm of response time of hardware IPs and a hardware IP auto-selection algorithm. The calculation algorithm can automatically calculate the response time of hardware IPs. The hardware IP auto-selection algorithm can select the suitable IP for the input application from some IPs having the same functions but different performance.

This paper is organized as follows. Section II defines a architecture of IP-based SoC. Section III proposes a processor core cosynthesis system which is the key issue in the proposed system. Section IV shows several experimental results compared with existing processors. Section V gives concluding remarks.

II. Target Architecture

Figure 2 shows an architecture model of IP-based SoC. The architecture is consisted of an processor core, a mem-

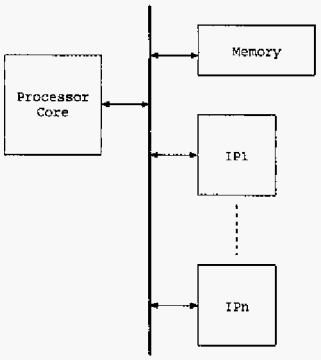

Fig. 2. An architecture model of IP-based SoC .

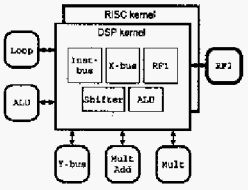

Fig. 3. An architecture model of processor core .

ory and several hardware IPs which are connected each other via a shared bus. Our approach is first the input application is partitioned into hardware/software parts, then the hardware parts are implemented by hardware IPs, and the software parts are implemented by a processor core.

In the following section, we define the processor core, the hardware IP and the interface of both units.

A. Processor Core

Figure 3 shows an architecture of the VLIW-type processor core which is consisted of a processor kernel and some optional hardware units. The architecture is based on [6].

A Processor kernel is (1) a RISC-type kernel or (2) a DSP-type kernel. A RISC-type kernel has the five pipeline stages composed of IF (instruction fetch), ID (instruction decode), EXE (execution), MEM (memory access) and WB (write back) stages. A DSP-type kernel has the three pipeline stages composed of IF, ID and EXE stages. The processor core can add the optional hardware units, such as functional unit (ALU, multiplier and so on) and addressing unit shown in Figure 3, to the processor kernels.

The processor core has basic instructions, parallel instructions and hardware-IP-instructions. The basic instructions are based on a general digital signal processor [4]. The parallel instruction executes more than one basic instructions. A processor core synthesis system determines which combination of basic instructions should be a parallel instruction based on an application program. The hardware-IP-instructions are described in the following section.

B. Interface

The interface between processor core and hardware IP is based on ARM Coprocessor Interface [5]. The ARM Coprocessor Interface defines signal interface and instruction interface.

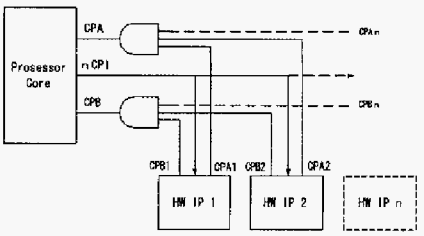

Fig. 4. A connection between processor core and hardware IPs .

B.1 Signal Interface

Figure 4 shows an connection of the processor core and the hardware IPs. The processor core can connect up to 16 hardware IPs. The processor core communicates with hardware IPs via three handshake signals as follows:

1. nCPI (Processor → IPs) A processor core wants to execute hardware-IP-instruction.
2. CPA (IP → Processor) There are no hardware IPs which can execute the hardware-IP-instruction.
3. CPB (IP → Processor) Hardware IP can not execute the hardware-IP-instruction immediately since the it is executing another hardware-IP-instruction.

B.2 Instruction Interface

Processor core sends hardware IPs three hardware-IP-instructions: (a) CDP (Coprocessor Data Operation), (b) LDC/STC (Coprocessor load and store operations) and (c) MCR/MRC (Register transfer operations). The format of hardware-IP-instruction is as follows:

```
CDP HW#, OP#
LDC HW#, N, Rd, Rn, offset
STC HW#, N, Rd, Rn, offset
MRC HW#, Rd1, Rd2
MCR HW#, Rd1, Rd2
```

CDP performs processing operation on the data held in the hardware IP's register. Each hardware IP is numbered and a processor core defines one HW IP to which it wants to send an CDP instruction by `HW#`. `OP#` shows which one of the operations the hardware IP have should be executed.

LDC/STC transfer data between a hardware IP and memory.

MCR/MRC transfer data between a processor core register and a hardware IP register.

C. hardware IP

Figure 5 shows an architecture of a hardware IP. It has a detapath, some registers, an instruction pipeline to hold the hardware-IP-instructions from the processor core and an instruction decode logic to decode the hardware-IP-instructions.

III. Processor core synthesis System

Figure 6 shows the proposed processor core synthesis system. Given an application program written in the

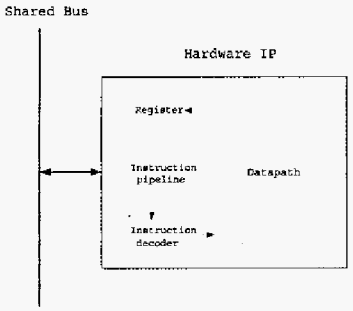

Fig. 5. An architecture model of a hardware IP .

SystemC language [3] and time constraint, the application is partitioned into hardware/software parts as shown in Figure 1. The CWL (Component Wrapper Language) [8] descriptions of hardware IPs which have the functions of hardware parts are also the input of the system. The system synthesizes a hardware description of a processor core and generates object code for the processor core and selected hardware IP's data.

Our approach is as follows:

1. **Compile:** First the system assumes a processor core to which all the hardware units are added and runs an application program on the assumed processor core. The compiler generates an assembly code which is minimum execution time and maximum area since there are no limitations of hardware units [6].
2. **Processor core HW/SW partitioning:** Second, the system replaces a part of hardware with software by eliminating hardware units added to a processor kernel one by one. The execution time of the assembly code becomes longer but the required area for processor core to run it becomes smaller [6].
3. The system repeats process 2 while the execution time of the assembly code satisfies the timing constraint and obtains a processor core satisfying the timing constraint with a small area.

We explain the consistents of the system.

response time calculator calculates the response time of hardware IP from the input CWL description.

pre-processor (SW extractor) extracts a SW description of an application from the system-level description.

profiler profiles software parts of the application from the system-level description.

compiler assumes a processor core to which all the hardware units are added and runs an application program on the assumed processor core. The compiler generates an assembly code. This assembly code is minimum execution time and maximum area since there are no limitations of hardware units.

processor core HW/SW partitioner replaces a part of hardware with software by eliminating hardware units added to a processor kernel one by one. The execution time of the assembly code becomes longer but the required area for processor core to run it becomes smaller.

HW IP auto-selector automatically selects the suitable hardware IP for the application from several hardware IP candidates which have the same functions but have different performances.

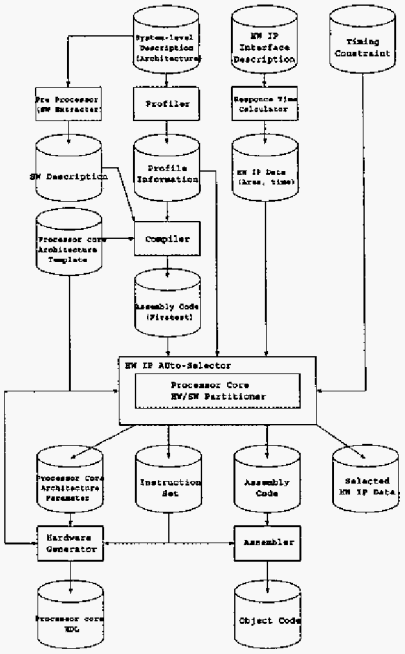

Fig. 6. A processor core synthesis system .

hardware generator generates a hardware description of the processor core.

assembler generates the object code of the application program run on the processor core.

See [6] for the compiler and the processor core HW/SW partitioner, and see [7] for the hardware generator. The pre-processor and the assembler are simple ones. We focus on the response time calculator and the HW IP auto-selector in the following section.

A. response time calculator

We propose an algorithm to calculate the response time of hardware IP from the CWL, the interface description language. We define the response time as; after the hardware IP receives the hardware-IP-instruction from the processor core, the response time is the time consumed by a hardware IP to execute a HW-IP-instruction.

CWL is a language used to define the interface specifications of the target IP correctly. Such interface specifications include specifications of logical signal changes as well as structural specifications, such as I/O pin information.

The CWL is consisted by four major sections; `port`, `alphabet`, `word`, `sentence`. Our algorithm focuses on `word` section.

Figure 7 shows the calculation flow. The `word` section defines the pattern of each transaction in normal representation. We classifies the `word` section into four expressions.

Basic expression is the most frequently used, consisted of `alphabet` and regular expressions.

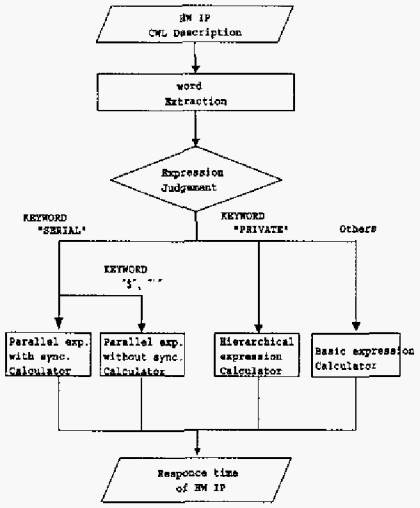

Fig. 7. A response time calculation flow.

Hierarchical expression represents the `word` hierarchically.

Parallel expression without synchronization represents the parallel processing such as pipeline.

Parallel expression with synchronization represents the parallel processing including synchronization.

In the response time calculator, these four expressions are applied to the different algorithms separately as shown in Figure 7.

Figure 8 shows an example of the response time calculation algorithm. The `word` section is extracted from the CWL description. The example is a parallel expression without synchronization, shown in the leftmost box in Figure 7 since there is the keyword "SERIAL".

A calculation algorithm for parallel expression without synchronization is:

1. Calculate the clock cycles of "SERIAL". "SERIAL" is consisted of the `alphabet` and regular expressions. We sum up all the alphabet taking account of regular expressions. In Figure 8, since all the alphabet (`alphabet a, b, c`) is counted as one cycle, `readcommand` is calculated as:

$$\begin{aligned} \texttt{readcommand} &= 1 \times 8 + 1 \times 3 + 1 \\ &= 12 \end{aligned} \quad (1)$$

   The basic expression is calculated in the same way as equation (1).

2. Calculate the response time of "PARALLEL". "PARALLEL" is consisted of `SERIAL words`, alphabet and parallel notation shown in Table I. The `readtransaction` is calculated as shown in Figure 8. We calculate the `PARALLEL word` from the last consistent with taking account of parallel notations as shown in table I. In Figure 8, the last consistent `parityerror` takes 8 cycles and the next one `readparity` takes 14 cycles. The relation between both consistents is "`&`" which means `parityerror` starts at the same time that `readparity` starts and the response time is calculated as:

$$\max\{\texttt{parityerror}, \texttt{readparity}\} = 14 \quad (2)$$

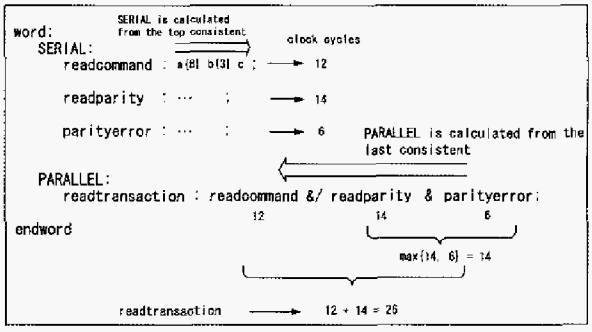

Fig. 8. An example of the response time calculator.

TABLE I
PARALLEL NOTATIONS AND CLOCK CYCLES.

Notation	Meaning	Cycles
a#b	"b" starts at the same time that "a" ends.	x + y
a#/b	"b" starts after "a" ends.	-
a&b	"b" starts at the same time that "a" starts.	max{x, y}
a&/b	"b" starts at after "a" starts.	x + y

Repeat this calculation to the top consistent and we can obtain the response time of the `readtsansaction`.

The proposed algorithms calculate the maximum clock cycles it could take. Therefore, it can calculate more precisely if the processing time does not rely on the quality of the data. And the more the designer writes the CWL in detail, the more the calculation result is precise.

B. HW IP Auto-selector

We also propose a hardware IP auto-selector which selects the suitable one from several hardware IPs having the same functions but different performances. The objective is to minimize the area of synthesized processor core and hardware IPs. The input of the hardware IP auto-selector is;

1. a timing constraint of the application,
2. the firstest assembly code obtained from the compiler,
3. profile information from the profiler
4. the data of reusable hardware IPs (area and response time obtained from the response time calculator).

The configuration of the processor core obtained from the processor core HW/SW partitioner varies by the hardware IP. Hardware IP auto-selector feeds back the configuration of the processor core to the processor core HW/SW partitioner and finds the optimal configuration. Hence we propose the algorithm which makes the number of trial minimum.

Figure 9 shows the flow of the hardware IP auto-selector. The selector reduces the hardware IP candidates from the hardware IP database by a candidate reduction algorithm. Then it selects one hardware IP from the candidates by a selection algorithm.

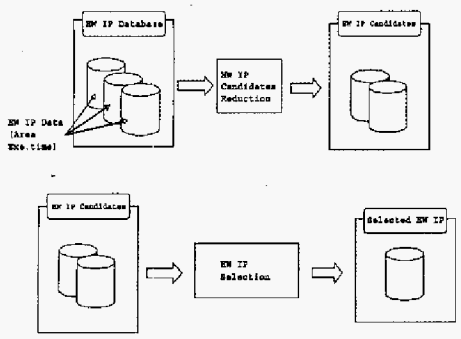

Fig. 9. A flow of the hardware IP auto-selector .

Candidates Reduction Algorithm The hardware IP auto-selector obtains the number of hardware IP instructions from the input profile information. Let us consider there are several hardware IPs which have the same functions. i-th hardware IP includes n operations. Let k be one of the n operations. Since the total processing time of the hardware IP is response time $\times$ number of instructions, the total processing time of i-th hardware IP (S_i) is defined as:

$$S_i = \sum_k T_{ik} I_{ik}, \tag{3}$$

where T_{ik} is a response time of k-th operation in i-th hardware IP. I_{ik} is the total number of k-th operations in i-th hardware IP executed in the application.

The auto-selector arranges all the hardware IPs in ascending order of the area and numbers them. It arranges all combinations of the hardware IPs if the designer decides to use two or more hardware IPs. S_i given by the equation (3) is expected to be in reverse. if the two hardware IPs, j and $j-1$, have the relation $S_{j-1} < S_j$, then jth hardware IP is eliminated.

Selection Algorithm After the hardware IPs are eliminated by the reduction algorithm, The auto-selector searches minimum combination of processor core and hardware IPs.

The hardware IP candidates have the trade-off between area and performance. The selection algorithm searches the minimum hardware IP (combination) which satisfies the timing constraint by the binary search. The key of the search is the area of hardware IP (total area of the combination).

Figure 10 shows the selection algorithm. The auto-selector selects the suitable hardware IP (combination) by $\log_2 n$ trials if the number of hardware IP candidate is n.

IV. Experimental Result

In the expriment, we use two application examples: the *JPEG Encoder* and the *3D-Animation*.

They were paritioned into hardware/software part. The hardware part was DCT in the JPEG encoder and hardware T&L in the 3D-animation. Then we applied the proposed processor core synthesis system to these applications.

Input: a timing constraint, assembly code obtained from compiler, profile information and data of hardware IP candidate
Output: assembly code, processor core architecture, instruction set and selected hardware IP.

Step1–1. Calculate the total area of all combinations of hardware IP candidates and arrange them in ascending order.
Step1–2. Let x_i be the total area of i-th combination (C_i), n be the number of combination and c_k be the target combination. The initial k is $n/2$.
Step1–3. Let t be the number of trial and the initial t is 0. The minimum total area of processor core and combination of hardware IPs is a in the trial process. and let r be the data (assembly code, processor core architecture, instruction set and data of hardware IP) of this trial.
Step2. Add 1 to the t. If $t > \log_2 n$, go to Step3. input the data of c_k with timing constraint and execution profile to the processor core HW/SW partitioner.

1. If no solution which satisfies the timing constraint is obtained, replace $k = k + \frac{n}{2^t}$ and repeat Step2.
2. If the solution which satisfies the timing constraint is obtained, let y_t the sum of the area of the obtained processor core and x_k, and
   (a) if $t = 1$ or $y_t < a$, replace k and a, $k = k - \frac{n}{2^t}$, $a = y_t$ respectively and repeat Step2.
   (b) if $y_t > a$, replace k $k = k + \frac{n}{2^t}$ and repeat Step2.

Step3. Output r and finish searching .

Fig. 10. A hardware IP selection algorithm .

A. JPEG Encoder

We gave some time constraints to the application. Table II shows the area, execution time and hardware configurations of the synthesized processor core.

The JPEG encoding system is consists of four major parts:

- Image Fragmentation
- DCT (Discrete Cosine Transform)
- Quantization
- Huffman Coding

In the design frame work shown in Figure 1, We partitioned off these parts into hardware/software parts and decided DCT to be implemented by hardware IP. We used Xilinx [9] 2-D DCT as a hardware IP. The response time of the hardware IP obtained by the calculation algorithm was 285 cycles.

B. 3D-Animation

We prepared six types of the T&L hardware IP shown in Table III in order to validate the effectiveness of hardware-IP- selector.

Table IV shows the area, execution time and hardware configurations of the synthesized processor core in some time constraints.

C. Discussion

In order to compare our results with the general purpose processors, We prepare the RISC processor having 5 stage pipelines, 32 32-bit registers, 32-bit instruction sets and the multiplier. We applied this processor core to the target architecture shown in Figure 2.

TABLE II
AREA, EXECUTION TIME AND HARDWARE CONFIGURATIONS OF SYNTHESIZED PROCESSOR CORE (JPEG ENCODER).

Timing consts [ms]	Execution time [ms]	Area [μm^2]	Hardware configuration		
			Kernel	#ALUs	#Regs
120.0	113.740	4,106,896	RISC	ALU*2, Mult*2	20
130.0	129.799	2,298,954	RISC	ALU*1, Mult*1	5
150.0	146.212	1,758,458	RISC	ALU*1, Mult*1	6
170.0	169.273	1,623,141	DSP	ALU*1, Mult*1	6
175.0	174.058	1,582,719	DSP	ALU*1, Mult*1	5

TABLE IV
AREA, EXECUTION TIME AND HARDWARE CONFIGURATIONS OF SYNTHESIZED PROCESSOR CORE (3D-ANIMATION).

Timig consts [ns]	Execution time [ns]	System area [mm^2]	Processor core area [mm^2]	Hardware configuration			HW IP
				Kernel	#ALUs	#Regs	
24.0	23.751	40.188014	4.795864	RISC	ALU*2,Mult*3	19	B
28.0	27.491	21.945339	2.820687	RISC	ALU*1,Mult*1	12	A
30.0	29.852	21.878155	2.753503	RISC	ALU*1,Mult*1	11	A
34.0	33.788	21.217409	2.092757	RISC	ALU*1,Mult*1	13	A
40.0	39.117	21.122514	1.997862	DSP	ALU*1,Mult*1	14	A

TABLE III
THE SPECIFICATION OF T&L HARDWARE IPS.

Name	Area [mm^2]	response time [$cycles$]
A	19.124652	98
B	35.392150	50
C	51.659648	38
D	67.927146	37
E	100.462142	34
F	198.067130	29

TABLE V
THE AREA AND EXECUTION TIME OF GENERAL PURPOSE RISC PROCESSOR.

Application	Area[μm^2]	Execution time[ms]
JPEG encoder	2,107,831	225.621
3D animation	2,107,831	41.829[ns]

Table V shows the area and the execution time of processor cores. In the 3D-animation, we use the Hardware IP **A** in the Figure III.

In Tables II and IV, our results show that compared with the general purpose processor core, the execution time of JPEG encoder was reduced 42%, and that of 3D-animation was reduced 20% though the area was almost the same.

In Table IV, Hardware IP **A** was selected in most case. Because the synthesized processor core and hardware IP can run in parallel, high performance of the hardware IP is not required for all the applications. Under the severe timing constraints, Hardware IP **B** was selected. Therefore, the hardware IP selector selects the suitable hardware IP up to the timing constraint of the application.

Overall the experimental results demonstrate that our processor core synthesis system effectively generates synthesizable processor cores based on application programs.

V. CONCLUSION

This paper proposed a processor core synthesis system based on the IP-based SoC. This paper also proposed two key issues for the system: hardware IP response time calculation algorithm and hardware IP auto-selection algorithm. The experimental results demonstrate that the system synthesizes processor cores effectively according to the features of an application program and synthesized processor cores have higher performances compared with general purpose processors.

The future work is the interface of processor core and hardware IPs. It is difficult to find the hardware IP defined in the section II.C. We will build a interface synthesis system for the proposed architecture.

ACKNOWLEDGEMENTS

We would like to thank Dr.Kei Suzuki and Mr.Hiroshi Ara at Hitachi, Ltd., Central Research Laboratory for many interesting discussions and suggestions which helped shape this paper.

REFERENCES

[1] I. J. Huang and A. M. Despain, "Synthesis of instruction sets for pipelined microprocessors," in *Proc. 31st DAC*, pp. 5–11, 1994.

[2] J. A. Rowson and A. Sangiovanni-Vincentelli, "Interface-Based Design," in *Proceedings of the Design Automation Conference*, pp. 178–183, June 1997.

[3] http://www.systemc.org/

[4] NEC, http://www.ic.nec.co.jp/micro/micro.html

[5] http://www.arm.com/

[6] N. Togawa, M. Yanagisawa, and T. Ohtsuki, " A hardware/software cosynthesis system for digital signal processor cores," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E82–A, no.11, 1999.

[7] M. Hamabe, A. Nose, N. Togawa, M. Yanagisawa and T. Ohtsuki, "A generation system for hardware description of pipelined processors," *Technical Report Of IEICE*, VLD97–117, ICD97–222, 1998 (in Japanese).

[8] Hitachi, Ltd., `http://koigakubo.hitachi.co.jp/~sl/cwl/html/index.htm`.

[9] Xilinx Inc., "2-D Discrete Cosine Transform v2.0," `http://www.xilinx.com/`.