An SoC Architecture and its Design Methodology using Unifunctional Heterogeneous Processor Array

Yoichi YUYAMA†, Masao ARAMOTO†,
Kazutoshi KOBAYASHI†† and Hidetoshi ONODERA†
†Graduate School of Informatics, Kyoto University, Kyoto 606-8501, Japan
{yuyama, masao, takai, onodera}@vlsi.kuee.kyoto-u.ac.jp
††VLSI Design and Education Center, The University of Tokyo, Tokyo 113-8656, Japan
kobayasi@ieee.org

ABSTRACT

We propose a heterogeneous processor architecture and its design methodology to shorten the design period of the SoC. It enables fast implementation of a system LSI including an embedded CPU and peripheral functional blocks. Each functional block of the system under design is implemented to a customized processor, instead of a peripheral hardwired logic. We customize processors by deleting unneccesarry functionalities, without adding new features. This eables rapid and bug-free design. Although area, power and performance of the proposed architecture are a little bit inferior to those of hardwired logics, the design period of the processor is considerably minimized, since the ROM pattern (software) and the layout pattern (customized processor, i.e. hardware) can be independently designed in parallel.

1. INTRODUCTION

According to the process minimization, huge number of transistors are integrated on a single die. On the other hand, it is very important to shorten the time to market. We have to develop efficient design methodologies for SoCs.

In order to accelerate the design period of SoCs, several system–level design languages are proposed such as SystemC [1]or SpecC [2]. Most of these languages are based on C/C++ language. C/C++ is widely used in the software development, so there are much of reusable design properties. Behavioral synthesis is going to be used to develop commercial LSIs, where untimed behavioral descriptions are synthesized to timed RTL or gate-level. Many of behavioral-level hardware modeling methods have been proposed [3–7].

However, in almost all behavioral synthesis systems, it is very hard to synthesize pointers, recursive function calls and so on that are frequently used in software design. According to process minimization, it takes an awful long time for a physical design after behavioral synthesis. A physical implementation for the description must be re-created whenever the behavioral description is modified because of bugs or specification changes. Therefore, the physical design cannot be started until the RTL or behavioral description is fixed.

Embedded SoCs for mobile/low-power applications consist of a general-purpose embedded processor and peripheral hardwired accelerators that execute specific functions. In this paper, we propose an SoC architecture, in which hardware accelerators are implemented as "heterogeneous processors." Each specific function/ thread written in software is mapped to a customized processor, which can be fixed at the early stage of the design. If a designer finds bugs on the software, it does not influence on physical design of the processor, but a program on a ROM. Our customization approach is deleting unnecessary units without adding new hardware. It enables rapid and bug-free hardware design easily. Physical-level designers have plenty of time to optimize a physical layout built from the netlist of the processor. Compared to hardwired logic, processor based SoC is a little bit inferior in performance and power consumption. But processor based SoC dramatically shortens a design period. It is also an excellent advantage to correct bugs and change functionalities of the product after shipment.

This paper is organized as follows. Section 2 describes the differences with related work. We explain proposed design methodology in Sect. 3. Section 4 explains the architecture of the customized processor and its performance. Section 5 describes our proposed architecture optimization results for JPEG encoding system. Finally, we conclude this paper in Sect. 6.

2. RELATED WORK

As Application Specific Instruction set Processors(ASIP), PEAS–III [8, 9], Xtensa [10], etc. are proposed. Valen–C [11] is proposed as a variable-bit width processor. Almost all ASIPs consist of a processor core with built-in accelerators. It is necessary for the ASIP itself to operate the whole application, which includes system control, function for OS, etc. A certain level of versatility is required in ASIP. It becomes difficult to optimize the bit width of ALU, register, etc. in ASIP, because it is assumed to perform the whole function of application by a single ASIP. Therefore, compared with hardwired logic, it is greatly inferior to hardwired logics in respect of area or power consumption. The amount of energy consumption may become about 100 times larger than hardwired logics.

As for the instruction level parallelism (ILP), embedded processors have capabilities of executing 4 instructions in parallel at most. If more and more parallelism is required, independent functional blocks must work in parallel. MeP(Media Embedded Processor) [12] is a customizable processor core for embedded systems. It is capable of optimizing its configurations and appending application-specific extensions for specific applications.

In our proposed methodology, we divide an application program into some functional blocks. A system consists of two or more connected processors each of which is specialized to each specific function. Enhancing the degree of parallelism lowers operating frequency and the whole power consumption is reduced. In each processor, only a single function in the applications is executed. It is possible to effectively optimize the bit width of ALUs or registers and so on. As compared with the general homogeneous multiprocessor architecture, it is possible to reduce the overhead of area or power consumption remarkably.

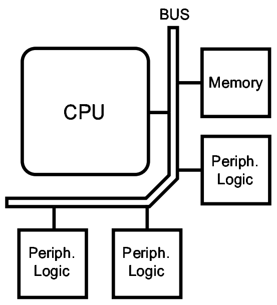


Figure 1: Conventional SoC architecture.

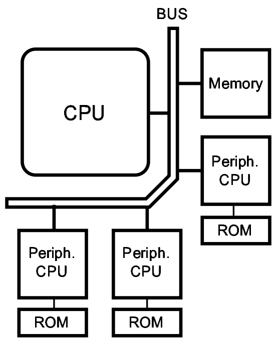


Figure 2: Proposed SoC architecture that consists of a main CPU and customized processors.

3. PROPOSED SYSTEM ARCHITECTURE AND DESIGN FLOW

3.1 Architecture of the Entire System

A conventional SoC architecture is shown in Fig. 1, in which peripheral accelerators are attached to a general-purpose processor through a bus. They work in parallel to accelerate a given work. Bottleneck functions are usually mapped to peripheral circuits. We have to design them on RTL or untimed behavioral level, which are very time-consuming tasks. Although behavioral synthesis shortens a logic design period than RTL, a physical design takes much longer time in the current sub-micron era. In order to correct a bug, it is necessary to go back to logic or behavioral level.

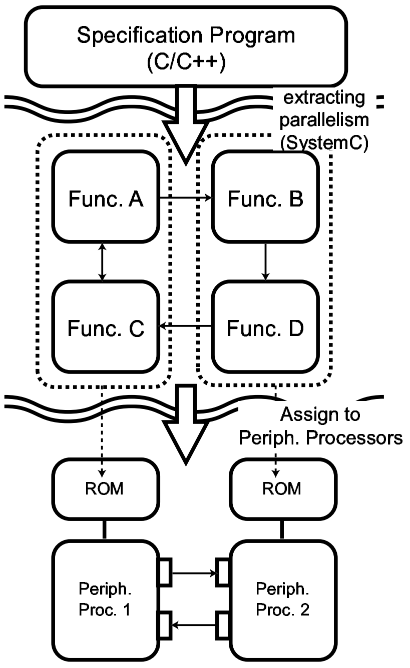
In the proposed methodology , SoC architecture is constituted as shown in Fig. 2. Peripheral circuits are implemented with processors specialized to each function instead of hardwired logics(Fig. 1).

If a general ASIP is used as a peripheral processor, its area and power consumption are unacceptable for embedded solutions, compared with those of hardwired logics. Our proposed method divides a system into a smaller block and it is assigned to a very small processor. We design a prototype of a customized processor, which is described in detail in Sect. 4.

3.2 Proposed Design Flow

The proposed SoC design flow is going on as follows.

1. Describe a specification program in the conventional C.
2. Rewrite the program in SystemC, in which a designer extracts parallelism to separate a program into thread/functional levels.
3. Fix organizations of customized processors to meet the specifications of all required functions.


Specification Program (C/C++)
extracting parallelism (SystemC)
Func. A
Func. B
Func. C
Func. D
Assign to Periph. Processors
ROM
ROM
Periph. Proc. 1
Periph. Proc. 2
Figure 3: Proposed design flow from a conventional C program to heterogeneous multiprocessors architecture.

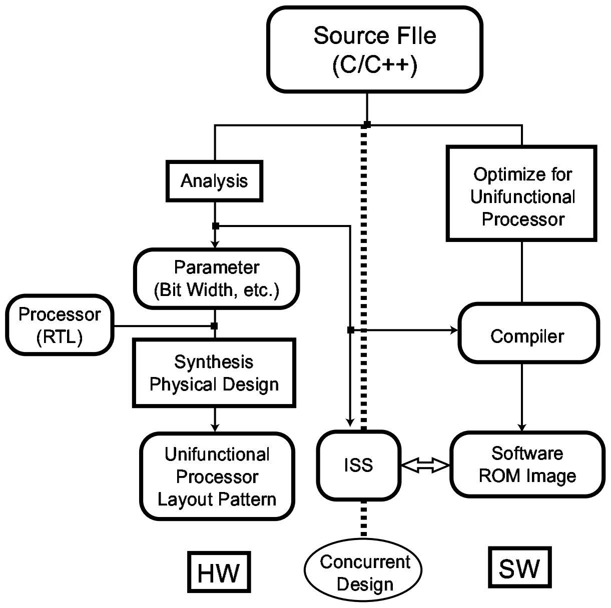
4. Physical designs of the processors and detailed design of software do in parallel

Fig. 3 shows how to extract parallelism from a specification program. First, we describe the program that fulfills specification in the conventional C. From the simulation of the specification program using ISS, the amount of processing time of each function in a program can be estimated. The specification program is divided into several functional blocks described in SystemC. SystemC is used only as a framework that describes parallelism. Each thread/-function is written with the conventional ANSI–C. Each functional block is mapped on a customized processor. Figure 4 shows the optimization flow of a functional block, which consists of customizable processor and its assigned program code. Simulation of the functional block by the ISS tells us required computing unit, bit width, execution time, and etc. Cumulative simulations by changing a parameter bring up the combination of the optimal parameters. From a processor description and the parameters, an RTL description for an optimized processor is generated and then its logic synthesis and physical design are going on. Each processor is customized by removing unnecessary units, instead of adding new hardware. This customizing method prevents adding a bug into the system.

While physical design is proceeding, software executed on the customized processor is going to be polished. Several trivial bugs of the software on those customized processors can be fixed by just replacing ROM patterns without changing physical layout of the processor.

4. ARCHITECTURE OF THE PROPOSED CUSTOMIZED PROCESSOR, AND ITS PERFORMANCE EVALUATION

In our proposed methodology, it is indispensable to make each processor as small as possible, since functions or threads are exe-


Source FIle (C/C++)
Analysis
Optimize for Unifunctional Processor
Parameter (Bit Width, etc.)
Processor (RTL)
Compiler
Synthesis Physical Design
Unifunctional Processor Layout Pattern
ISS
Software ROM Image
HW
Concurrent Design
SW
Figure 4: Process of the processor optimization.

cuted on multiple processors. This section explains the outline of the customized processor and the performance. We call it Minute Unifunctional Processor, abbreviated as MiU–Processor.

4.1 Architecture

The proposed customized processor has a subset of the SH-2 [13] instruction set. The instruction set of SH-2 processor has 62 categories, 142 instructions. In MiU–Processor, we have implemented 47 categories, 104 instructions except for system control, OS and so on. Its feature is as follows.

- Described in the SystemC RTL.
- Conventional 5-stage pipelined Harvard Architecture.
- Bit widths of the ALU and registers and number of registers are variable.
- Subset of an SH-2 instruction set.
- 23 instructions can be removed to minimize the area.

MiU–Processor has several different features compared with the original SH-2. Its memory architecture is the Harvard style Its register file has a capability to write data to a single register, while the original SH-2 has a dual-port register file. Thus the following change was given to the compiler in order to cope with these constraints.

- Post increment is forbidden.
- In the case of PC–relative addressing, data memory is accessed considering the value of PC is 0.
- Only delayed branch is supported.

It is described in SystemC RTL with several macro parameters for optimization. Its processor organization is varied according to giving parameters, such as bit width and/or the number of registers. Bit width of addition/subtraction, multiplication and logical operations can be changed independently. The instruction set of

Table 1: A result of logic synthesis and physical design of MiU–Processor.

Process	0.18μm CMOS Process 1–Poly, 5–Metal
Area	0.61mm^2
Max clk. Freq.	173MHz
Voltage	1.8V
Power	0.25 – 0.37mW/MHz

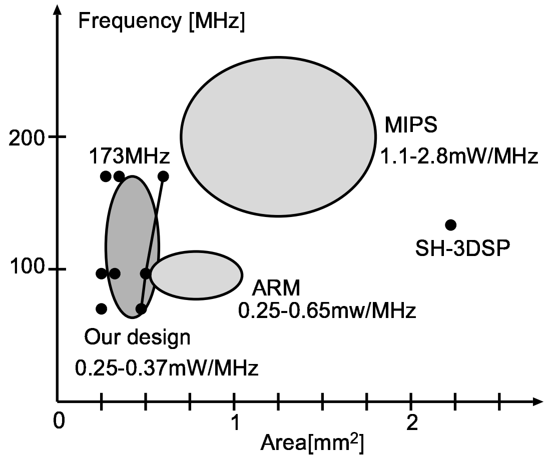


Figure 5: Comparison of area and operating frequency of processors.

the processor is divided into essential instructions, which cannot be removed, and the other removal 23 ones. We customize a processor by deleting unnecessary features from the template processor rather than adding features. So, it is not necessary to newly describe the hardware for acceleration, compiler, ISS and so on, which contributes to shorten verification time considerably.

4.2 Performance Evaluation of MiU–Processor

Table 1 describes a result of logic synthesis and physical design of MiU–Processor. We use a 0.18μm CMOS library [14].

Figure 5 shows area, operating frequency and power consumption of MiU–Processor compared with ARM [15], MIPS [16] and SH3-DSP [13]. In terms of an operating frequency, MiU–Processor is almost same as other processors. Its area is 0.49mm^2 at 70MHz, which is the smallest. Power consumption is 0.25 to 0.37 mW/MHz. This is pretty lower than MIPS, and almost equivalent to ARM.

4.3 Performance Variation by Changing Parameters of MiU–Processor

To evaluate area of the processor, it is synthesized to vary the following parameters: number of registers, organization of the ALU, and so on. Clock frequency is set to 166MHz.

Table 2 shows area of the processor by changing data-path width and number of general-purpose registers. It shows the area with all computing units. The processor with 32-bit data path is almost twice larger than that with 16-bit data path. Since register file occupies about 40 % of the processors in the case of 16 registers. The area is also reduced so as to reduce the number of general-purpose registers.

Table 3 shows areas by varying the organization of ALU. Data-path width is 32-bit and the number of general-purpose registers is

Table 2: Area(mm^2) by varying data path width and # of registers.

Data path width	# of registers		
	8	12	16
16	0.26	0.28	0.30
32	0.49	0.55	0.57

Table 3: Area(mm^2) by varying organization of ALU.

	Area
Minimum set	0.39
No multiplier	0.41
With 16-bit multiplier	0.51
Full set(includes 32-bit multiplier)	0.57

16. The minimum set in the table means no additional instructions, such as multiplication, multi-bit shift operations, string matching and so on. This result shows the bit width of the multiplier considerably affects the area. But the area is almost the same by adding the other instructions.

5. CASE STUDY : JPEG ENCODING SYSTEM

To evaluate the performance of MiU–Processor, we design a Motion JPEG encoding system with interconnected peripheral processors. It consists of the following blocks (processors) as in Fig. 6.

- YCC(Color Space Conversion:RGB to Y–Cb–Cr)
- DCT(Discrete Cosine Transform)/Quantization
- Huffman Encoding

All the peripheral processors are connected in series through FIFO.

The whole of this Motion JPEG encoding system is described in SystemC. SystemC is used only as a framework to describe parallelism. Inside each thread/function, conventional C descriptions can be used. Therefore, each functional module can be used not only as a program code cross–compiled to MiU–Processor, but also as a test bench for verification.

First, we compare performance of the customized processors with the circuits generated from behavioral synthesis. Note that both the C codes on the processor and used for the synthesis are produced from the same specification C programs. Here, YCC and DCT are used for comparison, since Huffman encoding seems to be too complex for behavioral synthesis.

The RTL description of the processor is synthesized through the Synopsys SystemC Compiler [6]. On the other hand, we use a behavioral synthesis tool called Bach from SHARP Corporation [3,

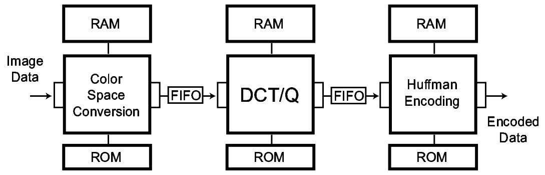


Figure 6: Block diagram of JPEG encoding system with Unifunctional Processor.

Table 4: Area and number of cycles of the customized processors and the hardwired logics from behavioral synthesis for processing a 16×16 macro block.

		Area (mm^2)	Power (mW)	# of cycle	Design Period (day)
YCC	MiU–Proc.	0.23	19.3	15,600	0.5
	HW logic	0.14	14.1	8,400	2.0
DCT	MiU–Proc.	0.28	23.3	20,500	1.0
	HW logic	0.30	15.3	8,900	3.5

17] to synthesize hardwired logics from behavioral descriptions. A 0.18μm CMOS library [14] is used for synthesis. Clock frequency is set to 166MHz.

Table 4 shows the number of cycles and area (including ROM/ RAM area). Note that the number of cycles is for processing a single macro block(16× 16). The areas of the processors are almost same or about twice larger than those of the hardwired logics. The processors consumes about half as larger power, while the number of cycles are about 2.5 times larger than the hardwired logics.

Next, we evaluate the design period, which means the time from algorithm C/C++ source codes to functional verification and power estimation. The procedure is as follows in our proposed method.

1. Parameter analysis from a specification program and logic synthesis of the processor
2. Software optimization
3. Functional verification, power estimation and so on

On the other hand, it becomes the following procedures in behavioral synthesis.

1. Modifying a specification program to be synthesizable by a behavioral synthesis tool
2. Behavioral synthesis and logic synthesis
3. Functional verification, power estimation and so on

As for the design and verification period, however, our processor needs only for several hours. On the other hand, it takes at least several days to design and evaluate circuits with behavioral synthesis. In the method using behavioral synthesis, it takes long in changing description and verifications. Changing description into synthesizable one may be difficult, like Huffman encoding. It must take several more days with RTL.

Table 5 shows performances of peripheral processors and whole Motion JPEG encoding system. Parameters such as data path width, number of registers and etc. of each processor are not optimized for each specific function. The Motion JPEG program itself is not optimized. The number of cycles in the table is for compressing a CIF(352×288) image by JPEG. Area and number of gates include size and number of gate of Memory. The area of ROM and RAM is converted to that of the NAND2 gate.

The number of gates is 31.8k, which is about 3/4 of the Motion JPEG LSI designed in RTL [18] in Table 7. Our system compress 8 frame/s at 150MHz, about 1/4 of performance compared with ASIC [18] that enables 30 frames/s at 18MHz. But it takes within one day to design whole system (Table 5) from the specification programs.

Then we optimize software programs, data path width, number of registers and the size of memory. Table 6 shows area, # of gates

Table 5: Area, # of gates, and etc. of peripheral processors in JPEG encoding system before optimization.

Function	Area (mm^2)	# of gates	# of cycle	Design period
YCC	0.61	10.4k	14.0M	<0.5day
DCT/Q	0.61	10.4k	18.4M	<0.5day
Huffman	0.63	11.0k	14.8M	0.5day
Total	1.85	31.8k	18.4M	1.0day

	Data path	# of registers	ROM	RAM
YCC	32bit	16	0.8kB	4.5kB
DCT/Q	32bit	16	1.8kB	0.3kB
Huffman	32bit	16	4.6kB	0.5kB
Total	–	–	7.2kB	5.3kB

Table 6: Area, # of gates, and etc. of peripheral processors in JPEG encoding system after optimization.

Function	Area (mm^2)	# of gates	# of cycle	Design period
YCC	0.23	4.7k	9.9M	0.5day
DCT/Q	0.28	5.7k	12.1M	1day
Huffman	0.63	11.0k	12.7M	1.5day
Total	1.14	21.4k	12.7M	3.0day

	Data path	# of registers	ROM	RAM
YCC	16bit	8	0.6kB	0.3kB
DCT/Q	16bit	16	1.5kB	0.2kB
Huffman	32bit	16	4.6kB	0.4kB
Total	–	–	6.7kB	0.9kB

and etc. of Motion JPEG encoding system, which consists of optimized MiU–Processors and software codes. The number of gates is 21.4k, which is 40% smaller than the system without optimization. It is because we reduce data path width of YCC and DCT/Q processors to 16 bits. The number of cycle is 20% smaller by optimizing program codes for the instruction set of each processor. Compared with hardwired logics [18], the number of gates is almost half. Our system compress 12 frame/s at 150MHz, about 1/3 of performance compared with [18]. By doubling the number of peripheral processors, its area becomes equal and the penalty of the cycles is reduced to half. Table 7 shows performance of Motion JPEG encoding system designed in RTL [18] and our proposed system. Figure 7 describes encoding time and number of gates in each design stage.

Here we discuss the design period. By our method, it is possible to complete the whole system design within one day. We accelerate the functions on each processor by polishing the software code and at the same time synthesize all the processors within another two days. Therefore, it takes only three days to complete the logic level design of the whole system.

6. CONCLUSION

We propose an SoC architecture, with "heterogeneous processors," in which functional blocks are implemented as customized processors, not as hardwired logics. We have implemented a customized processor core described in SystemC-RTL. It can be optimized to a set of given operations, which are conventionally implemented in specified hardwired logics. Although the area, power and performance of the processor are little bit inferior to those of hardwired logics, the processor is flexible to a trivial software modification. A system designer can fix the processor RTL code at the very early stage of the design flow from a scratch C code of the target function. When a software designer modifies the software code, it is influenced only in a ROM pattern of the processor, not in its physical design, which must be designed carefully to meet the design rule and the timing requirement.

We customize the processors for these three operations: RGB to Y–Cb–Cr conversion, Discrete Cosine Transform (DCT), Huffman encoding, which constitute a Motion JPEG encoding system. They are compared with hardwired logics from behavioral synthesis. Consequently, area and the number of cycles of each processor are just twice of those of the hardwired logic. About the whole Motion JPEG encoding system, our system compress 12 frame/s at 150MHz, about 1/3 of performance compared with circuit designed in RTL [18]. On the other hand, the design time of the proposed architecture is quite shorter than the hardwired logic, since the software (ROM pattern) and the hardware (layout pattern) can be designed simultaneously.

If there is a severe requirement for the given operations, hardwired logics must be used to meet the specifications. The proposed method may not be applied to data-intensive applications in which the data-level parallelism is very high such as motion estimation. But if the given operations contain so many threads or functions, each of which has very little data-level parallelism, the proposed design methodology is very effective to implement an LSI that meets a given specification in very short design period.

At the point of area and performance, a circuit by our proposed

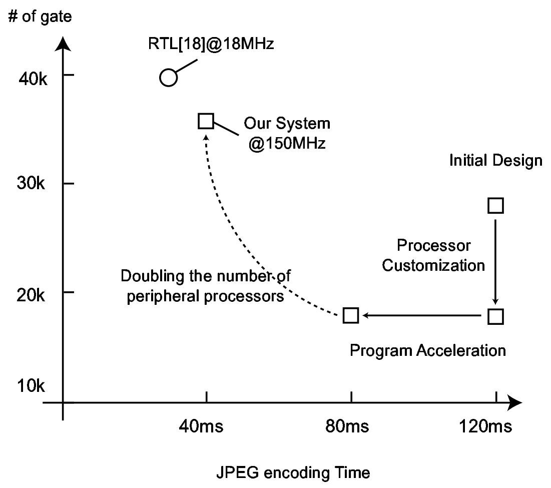


Figure 7: Performance of Motion JPEG encoding system using Unifunctional Processor.

Table 7: Performance of Motion JPEG encoding system designed in RTL [18] and Unifunctional Processor array

	Manual Design	MiU–Processor
# of gate(core)	40k	34.4k
RAM	2.1kB	1.8kB
Image size	VGA	CIF
Frame rate	30 fps(@18MHz)	24 fps(@150MHz)

method is a little bit inferior to those from hand–coded RTL and behavioral synthesis. But our method promotes rapid design. It is very effective in the first model of a product. In the LSI market, the profit from each chip is the highest at the early stage. Our proposed method fits the design of the first model. After it is launched to the market, each processor is gradually replaced to hardwired logic for performance, area, power and so on. With our proposed method, we can change system specification and functionality. These features enable bug correction and addition of functionality after product shipment. For example, by applying to networking equipment, it becomes possible to change a communication protocol or a security program.

7. REFERENCES

[1] Synopsys, Inc., Coware, Inc., Frontier Design, Inc. *SystemC Version 2.0 User's Guide*, 2001.

[2] Daniel Gajski Rainer Domer, Andreas Gerstlauer. *SpecC Language Reference Manual v1.0*, 2001.

[3] T. Kambe, A. Yamada, K. Nishida, K. Okada, M. Ohnishi, A. Kay, P. Boca, V. Zammit, and T. Nomura. A c-based synthesis system, bach, and its application. *Proceedings of the ASP-DAC 2001. Asia and South Pacific Design,no.78*, pages .151–5, 2001.

[4] Kazutoshi Wakabayashi. C-based synthesis experiences with a behavior synthesizer "cyber". In *DATE'99*, pages 390–393, March 1999.

[5] Coware, Inc. *CoWare N2C Datasheet*. http://www.coware.com/cowareN2C.html.

[6] Synopsys, Inc. *Synopsys CoCentric SystemC Compiler*. http://www.synopsys.com/products/cocentric_systemC/.

[7] Y.-L. Lin. Recent development in high level synthesis. *ACM Transactions on Design Automation of Electronic Systems*, 2(1):2–21, 1997.

[8] Makiko Itoh and Shigeaki Higaki. PEAS–III: An ASIP Design Environment. In *2000 IEEE INTERNATIONAL CONFERENCE ON Computer Design: VLSI in Computers & Processors*, pages 430–436, September 2000.

[9] Jun Sato, Masaharu Imai, Tetsuya Hakata, Alauddin Y. Alomary, and Nobuyuki Hikichi. An Integrated Design Environment for Application Specific Integrated Processor. In *1991 IEEE INTERNATIONAL CONFERENCE ON Computer Design: VLSI in Computers & Processors*, pages 414–417, October 1991.

[10] Tensilica, Inc. *Xtensa*. http://www.tensilica.com.

[11] F. N. Eko, A. Inoue, H. Tomiyama, and H. Yasuura. A Soft-Core Processor Architecture for Embedded System Design. In *Asia Pacific Conference on Hardware Description Languages*, pages 154–159, July 1998.

[12] MeP Media Embedded Processor. TOSHIBA, Inc. http://www.mepcore.com.

[13] Hitachi, Inc. *SuperH*. http://www.superh.com/.

[14] Hidetoshi Onodera, Masanori Hashimoto, and Tetsutaro Hashimoto. Asic design methodology with on-demand library generation. *Proc. Symposium on VLSI Circuits*, pages 57–60, 2001.

[15] ARM Ltd. *ARM7 THUMB FAMILY*. http://www.arm.com.

[16] MIPS Technologies, Inc. *MIPS32 K4 Family*. http://www.mips.com.

[17] Yoichi Yuyama, Kosuke Takai, Kazutoshi Kobayashi, and Hidetoshi Onodera. Hardware and Software Codesign with Using SystemC and Bach. In *DATE'02 Designers' Forum*, pages 30–34, March 2002.

[18] S. Okada, Y. Matsuda, T. Watanabe, and K. Kondo. A SINGLE CHIP MOTION JPEG CODEC LSI. In *IEEE Transactions on Consumer Electronics, Vol. 43, No. 3*, pages 418–422, August 1997.