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 unnecessary functionalities, without adding new features. This enables 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 may become about 100 times larger than hardwired logics.

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.

2. RELATED WORK

As Application Specific Instruction set Processors (ASIP), PEAS-III [8, 9], Xena [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.
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.
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-
executed 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 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 0.49mm² 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(mm²) by varying data path width and # of registers.

<table>
<thead>
<tr>
<th>Data path width</th>
<th># of registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0.26 0.28 0.30</td>
</tr>
<tr>
<td>16</td>
<td>0.49 0.55 0.57</td>
</tr>
</tbody>
</table>

Table 3: Area(mm²) by varying organization of ALU.

<table>
<thead>
<tr>
<th></th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum set</td>
<td>0.39</td>
</tr>
<tr>
<td>No multiplier</td>
<td>0.41</td>
</tr>
<tr>
<td>With 16-bit multiplier</td>
<td>0.51</td>
</tr>
<tr>
<td>Full set</td>
<td>0.57</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>Area (mm²)</th>
<th>Power (mW)</th>
<th># of cycle</th>
<th>Design Period (day)</th>
</tr>
</thead>
<tbody>
<tr>
<td>YCC</td>
<td>0.23</td>
<td>19.3</td>
<td>15,600</td>
<td>0.5</td>
</tr>
<tr>
<td>DCT</td>
<td>0.28</td>
<td>23.3</td>
<td>20,500</td>
<td>1.0</td>
</tr>
<tr>
<td>Full set(includes 32-bit multiplier)</td>
<td>0.57</td>
<td>15.3</td>
<td>8,900</td>
<td>3.5</td>
</tr>
</tbody>
</table>

Figure 6: Block diagram of JPEG encoding system with Unifunctional Processor.
Table 5: Area, # of gates, and etc. of peripheral processors in JPEG encoding system before optimization.

<table>
<thead>
<tr>
<th>Function</th>
<th>Area (mm²)</th>
<th># of gates</th>
<th># of cycle</th>
<th>Design period</th>
</tr>
</thead>
<tbody>
<tr>
<td>YCC</td>
<td>0.61</td>
<td>10.4k</td>
<td>14.0M</td>
<td>&lt;0.5day</td>
</tr>
<tr>
<td>DCT/Q</td>
<td>0.61</td>
<td>10.4k</td>
<td>18.4M</td>
<td>&lt;0.5day</td>
</tr>
<tr>
<td>Huffman</td>
<td>0.63</td>
<td>11.0k</td>
<td>14.8M</td>
<td>0.5day</td>
</tr>
<tr>
<td>Total</td>
<td>1.85</td>
<td>31.8k</td>
<td>18.4M</td>
<td>1.0day</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data path</th>
<th># of registers</th>
<th>ROM</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>YCC</td>
<td>32bit</td>
<td>16</td>
<td>0.8kB</td>
</tr>
<tr>
<td>DCT/Q</td>
<td>32bit</td>
<td>16</td>
<td>1.8kB</td>
</tr>
<tr>
<td>Huffman</td>
<td>32bit</td>
<td>16</td>
<td>4.6kB</td>
</tr>
<tr>
<td>Total</td>
<td>–</td>
<td>–</td>
<td>7.2kB</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Area (mm²)</th>
<th># of gates</th>
<th># of cycle</th>
<th>Design period</th>
</tr>
</thead>
<tbody>
<tr>
<td>YCC</td>
<td>0.23</td>
<td>4.7k</td>
<td>9.9M</td>
<td>0.5day</td>
</tr>
<tr>
<td>DCT/Q</td>
<td>0.28</td>
<td>5.7k</td>
<td>12.1M</td>
<td>1day</td>
</tr>
<tr>
<td>Huffman</td>
<td>0.63</td>
<td>11.0k</td>
<td>12.7M</td>
<td>1.5day</td>
</tr>
<tr>
<td>Total</td>
<td>1.14</td>
<td>21.4k</td>
<td>12.7M</td>
<td>3.0day</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data path</th>
<th># of registers</th>
<th>ROM</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>YCC</td>
<td>16bit</td>
<td>8</td>
<td>0.6kB</td>
</tr>
<tr>
<td>DCT/Q</td>
<td>16bit</td>
<td>16</td>
<td>1.5kB</td>
</tr>
<tr>
<td>Huffman</td>
<td>32bit</td>
<td>16</td>
<td>4.6kB</td>
</tr>
<tr>
<td>Total</td>
<td>–</td>
<td>–</td>
<td>6.7kB</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Manual Design</th>
<th>MiU-Processor</th>
</tr>
</thead>
<tbody>
<tr>
<td># of gate(core)</td>
<td>40k</td>
</tr>
<tr>
<td>RAM</td>
<td>2.1kB</td>
</tr>
<tr>
<td>Image size</td>
<td>VGA</td>
</tr>
<tr>
<td>Frame rate</td>
<td>30 fps(18MHz)</td>
</tr>
</tbody>
</table>

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
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
