Memoirs of the Osaka Institute
of Technology, Series A
Vol.48,No.1(2003) pp.31~46

A Design Methodology for Re-Configurable MPU for an Embedded System and Software on an FPGA*

by

Hideo ARAKI**, Toshiro KUTSUWA and Katsumi HARASHIMA

Department of Electronics, Information and Communication Engineering, Faculty of Engineering

(Manuscript received May 29, 2003)

Abstract

FPGA (Field Programmable Gate Array) is an essential device to design of hardware at present. Ways of applying FPGAs are studied actively for system design problems. The problems are on design methods for large scale and speed-up. One of the design methods is SoC (System on a Chip) that realizes to implement all functions of a system to a chip. To realize SoC efficiently, IP (Intellectual property) is very important and is developed for re-use of hardware. Many MPUs (Micro Processing Unit) are developed as IPs. However, most of the MPUs at present have too high specifications and too large scales, as applying to embedded systems. Furthermore, embedded MPUs require functions to execute real-time application program. Then we propose a flexible and small scale MPU and it's design method. This paper is shown our proposed design method with "specifications description language" and the examples to apply the design method, and indicates problems and reasons to implement processors in an FPGA.

* This paper was presented in ITC-CSCC2002 (July 16-19, 2002 in Phuket, Thailand) (The 2002 International Technical Conference on Circuits/Systems, Computers and Communications.)
** Major in Electrical and Electronics Engineering, Graduate Course of Engineering.

1. Introduction

At present, researches for applying FPGAs are very active. Furthermore the possibility of applying FPGAs is expanded with expansion of FPGAs capacity. The FPGA is applied for rapid-proto-typing, re-configurable-system and simplification of debugging. To realize a rapid-prototyping, the FPGA is used instead of a VLSI (Very Large Scale Integrated-circuit). In this case, designers decide the behavior of the functions, and build functions. M Gschwind et al. have researched to prototype a MPU using FPGA [1]. In the paper, they have built MIPS RISC processor described by VHDL for an FPGA and a VLSI. B. K. Tan et al. have researched for re-configurable digital signal processor for ASIC[2] (Application Specific Integrated Circuit). S. Honda et al. have researched for a co-design on device and device-driver using a "SpecC"[3]. The researches have following branches: "Description", "Synthesize", "Design Device" and so on. Most of the researches applying FPGA are for rapid-prototyping on a VLSI. However, it is difficult to apply the results of the researches for the embedded system, because these processors are too large.

In addition to them, some MPUs are produced by FPGA venders. The MPUs are able to modify a little only. Ability of the modifications is very important to fit for embedded systems which have a little hard ware resource. We think that a MPU which is full and free synthesis is the most suitable for embedded systems. Therefore a MPU to fit an embedded system needs the following characters; "Easy to modify a structure", "Easy to supplement functions" and "Small scale". To realize the MPU with the characters, we research a design method and a MPU.

At the first, we propose an "Implementing Style" to build a system with an FPGA. Several approaches for the implementing styles are tried as various applications. One of the implementing styles is to include a MPU in an FPGA. However, some approaches are tried and applied. Figure 1 illustrates the relations of an FPGA, functions and implemented method.

1)FPGA has a MPU as a hardware macro which cannot be changed, and other devices are connected on a PCB (Printed Circuit Board) (Fig.1 (a)).

2)FPGA has only ASIC functions. MPU and other devices are connected on a PCB. We can change the functions on an FPGA (Fig.1 (b)).

3)FPGA has a MPU as a software macro and ASIC functions. The MPU can be changed a little (Fig.1 (c)).

4)FPGA has a MPU and other devices as description with various functions. The MPU can be changed in full scale (Fig.1 (d)).

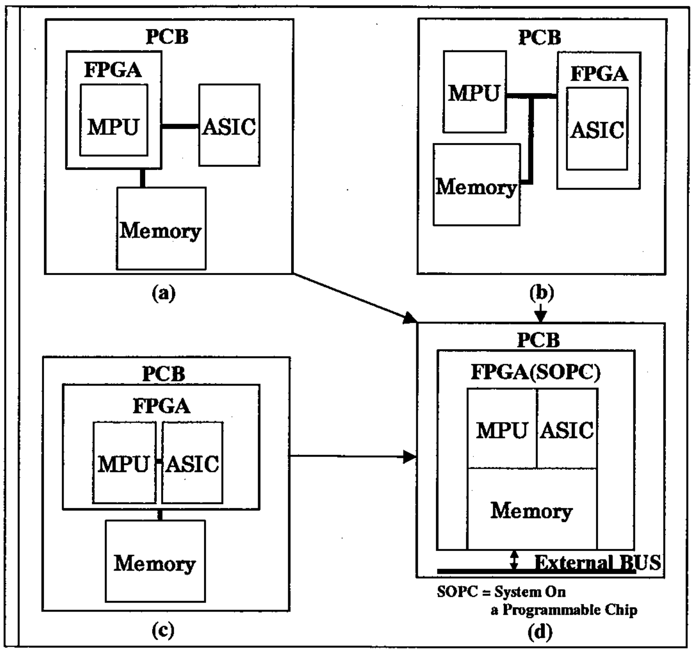


Fig.1 Difference of implementation.

We try to the 4th style (Fig.1 (d)), because a MPU on the style is most useful for system modifications. To apply the MPU for embedded systems, there are some advantages and some defects.

- We can build the most suitable MPU for the specified application. (advantage)
- We can use the hardware efficiently. (advantage)
- We can get a necessary speed. (advantage)

- To build the MPU is very difficult. (defect)
- No software development tool is exploited for the MPU. (defect)

To conquer the defects, we propose a design style with that specification description language expresses a MPU on an FPGA. The proposing includes following that.

- Describe the MPU and the other hardware by a specification description language.
- Describe the software for the MPU by the specification description language.
- Improve the system re-configuration by these descriptions.

2.Design Methodology

An evolution of an FPGA gives an FPGA a logic-cell (Basic Composition of an FPGA) based on SRAM technology. Furthermore, logic-cells for memory and random logic are mounted separately. An effective method of the characters for the separate logic-cells is that functional implementation is executed by not hardware-logic but software-logic. However, the implementation of the functions with a critical time restriction is executed effectively by hardware-logic than software-logic. Our proposing design method is applied to aim to use resources of the FPGA with random-logics and memory-cells effectively.

Descriptions about specifications decide the way of implementation by hardware or software. By using a specification description language, we can change easily the way, because the language has high flexibility to imprement them. To use the language is a good technique for designing a re-configurable system with an FPGA, because there are often several changes of specifications and requirement at designing a small-scale embedded system. Our proposing design method implements a re-configurable MPU in the FPGA, and applies it as a programmable controller. The MPU has structural modules, and the modules are able to change independently. The structure of the FPGA is efficiently and does not lose flexibility.

Furthermore, the FPGA has a some functional unit as a module which has a same style with the MPU. Figure.1 (d) shows the unit as a ASIC. The MPU and the ASIC communicate by special registers in the MPU each other. The register has asynchronous single direction ports, and is like a FIFO (First In First Out) buffer. The MPU accedes to the ASIC accesses to the register without a bus-arbitration. In addition to that, the ASIC can connect to internal bus of MPU by necessity. Our method can select a communicating way from the MPU to the ASIC, considering with a tradeoff between forwarding speed and transaction speed.

To convert software and hardware efficiently each other, applying the specification description language and the style of the module's description are very important. Furthermore, the MPU can be modified in our proposed design method. The MPU that is re-configurable structure and expressed by specification description language has following characteristics.

1)By modifying a command decoder block, the hardware that is specified for applications can be controlled by an additional command from software.

2)By adding some ALU blocks, the MPU can execute parallel calculations.

3)The MPU can have a specific calculation unit by modifying an ALU.

4)External units can access the register file directly.

5)External units can access the program counter unit directly.

The characteristics of the MPU can realize an easy and quick design for a flexible system. Besides, the MPU can be applied and extended more flexibly than other MPUs that are distributed as an IP. A description of program codes for the MPU can be modified uniformly, because the codes and the MPU are written by the same specification description language. By memorizing the codes on the FPGA, only one configuration file of the FPGA is necessary for the system. Therefore, only one controller can be found on the system.

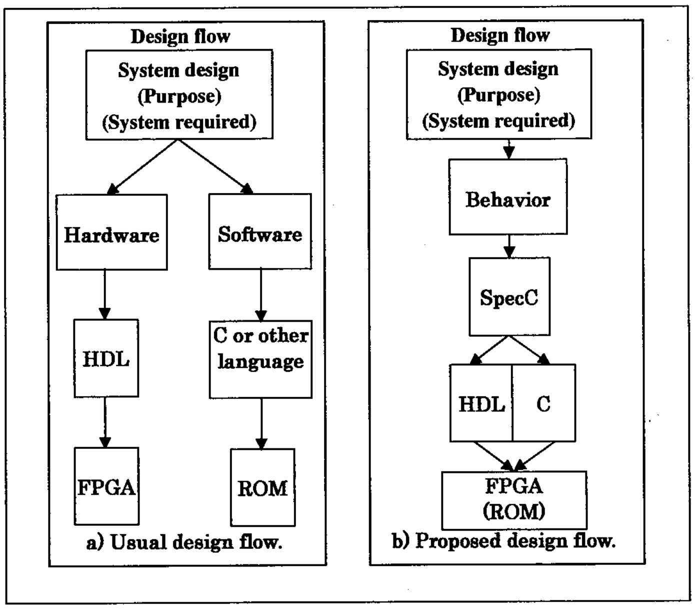


Fig. 2 Design flow.

We propose to use "SpecC" which is a specification description language to realize this method[4)]. By applying "SpecC", a design of the system can be changed by the request as a minimum modification. "SpecC" is based on "ANSI C language". To evaluate a behavior of the system, we can use "C language code" that is generated from "SpecC language code". Figure 2 illustrates difference between a usual design flow and our proposing design flow. The proposal can ensure the flexible co-design of hardware and software. The flexibility is very important to design a re-configurable system. The codes that are described in "SpecC" for the re-configurable system are divided to HDL (Hardware Description Language) codes and C program codes. Each source codes generate each object file by each compiler. Finally, an object file for programming to FPGA is generated from all object files.

We develop a computer simulator based on this design technique, and evaluate the system's behaviors and the MPU's efficiency in experiments with the simulator.

3.Experiments and Results

3.1 Build the MPU and Application System

At the first, we evaluate about the description style. We design an experiment system with an 8Bit MPU and a motor control unit, and generate an experiment system simulator from the description.

The experiment system performs following operations.

(1) A user specifies a speed from general-purpose input port on the FPGA.

(2) The speed is detected by measuring electromotive force of the motor.

(3) The control of the motor is performed so that specified speed and a measuring speed is equal to the specific speed.

(4) The motor speed is controlled as a motion with constant angle acceleration.

Our MPU has suitable design configuring to FPGA, and its inside is composed of function modules. The MPU has following function modules; Command-Decoder, Command-Controller, Register File, Program Counter, Stack Unit, ALU (Arithmetic Logic Unit), Command Bus, and Data Bus. Figure 3 illustrates a block diagram in the MPU. Figure 4 illustrates a block diagram as function modules.

・Command Decoder reads a program code on a Program Memory at the address indicated by Program Counter through the Command Bus.

・Command Decoder analyzes the program code.

・Command Controller controls the other modules according to instruction of Command Decoder.

・Register File write a datum on Data Bus to a register, or output a datum on a register to Data Bus, according to Command Controller.

・Program Counter increases a value in PC-register (Program Counter register), or sets a value on Data Bus to PC-register, according to Command Controller.

・A stack unit in Program Counter records the value of PC-register according to Command Controller.

・ALU outputs a calculated value to Data Bus.

・ Data Bus is connected to Register File, ALU, and Program Counter. These modules exchange values through the Data Bus.

・The motor control unit which is in the ASIC controls speed by PWM. This control unit is connected to Register File of the MPU, and it controls a motor in non-synchronism.

The MPU can access the control unit through three registers as Speed Control Register, Timer Register and Setup Register. A value written into the registers is reflected to a controller on real time except for the timer register. The motor control unit has an 8bit input port. A value of the input port is reflected to the register of the MPU on real time.

- Additionally, MPU has general-purpose I/O ports of 8bit. The I/O ports are connected to the register of MPU directly.

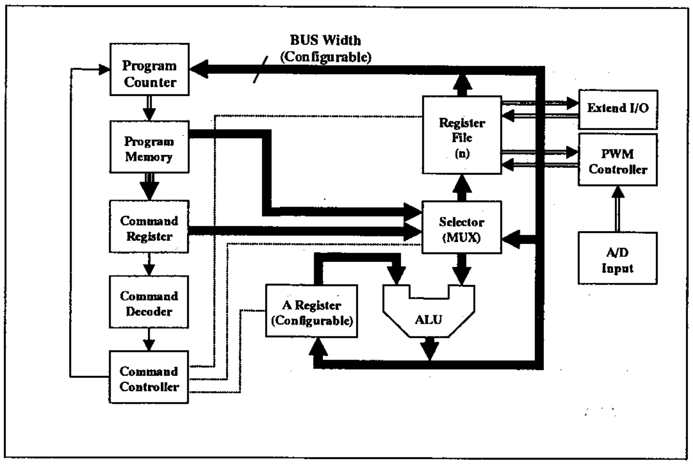


Fig. 3 MPU block for experiment.

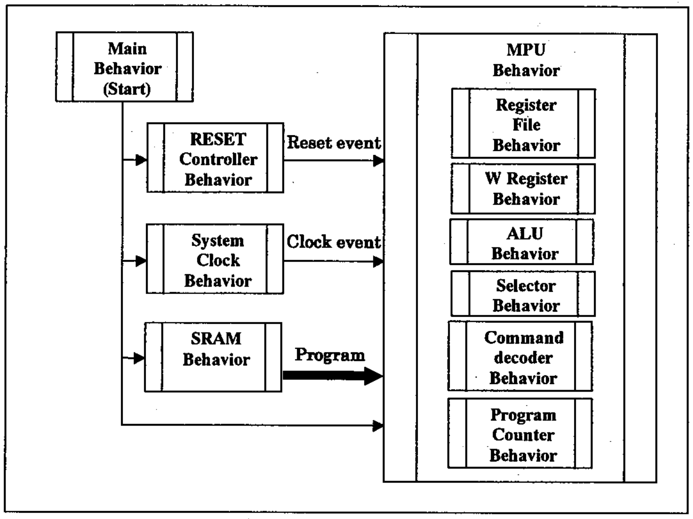


Fig. 4 Function modules in MPU.

Next, implementing software to the system is shown. Figure 5 shows the general flow chart for the software. The software's movements are as follows.

(1) All registers are initialized.

(2) The MPU reads the value of specified speed (Ws) from general-purpose I/O, and sets a value to the MPU's register.

(3) The MPU reads the value of detected speed from motor control unit, and sets a value to the MPU's register (Wd).

(4) PWM is controlled by the following rules.

In case of (Wd-Ws) > 0 (acceleration)

(Wd-Ws) >=8 (angular acceleration increase by 1)

(Wd-Ws) <8 (angular acceleration increase by (Wd-Ws)/2)

In case of (Wd-Ws) = 0 (fixed speed): No action.

In case of (Wd-Ws) < 0 (deceleration)

(Wd-Ws) >-8 (angular acceleration decrease by (Wd-Ws)/2)

(Wd-Ws) <=-8 (angular acceleration decrease by 1)

(5) MPU calculates the angular acceleration (Aw).

(6) MPU writes the value of (Wd+Aw) to PWM register (Wo).

(7) Repeat (2) to (6).

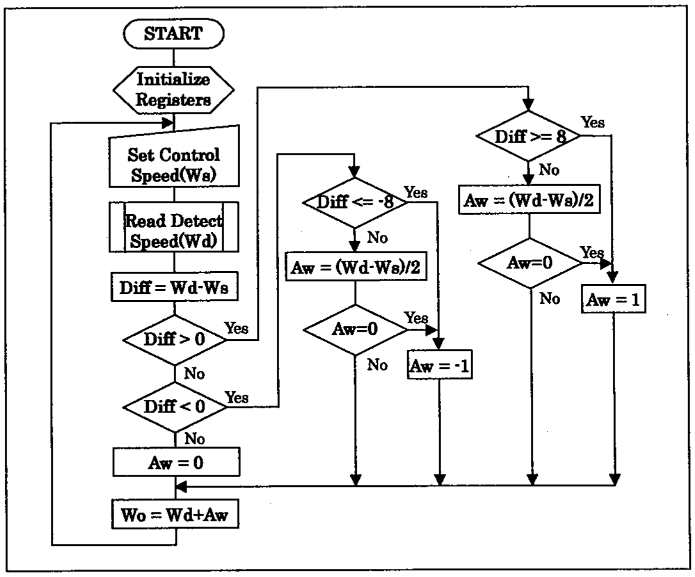
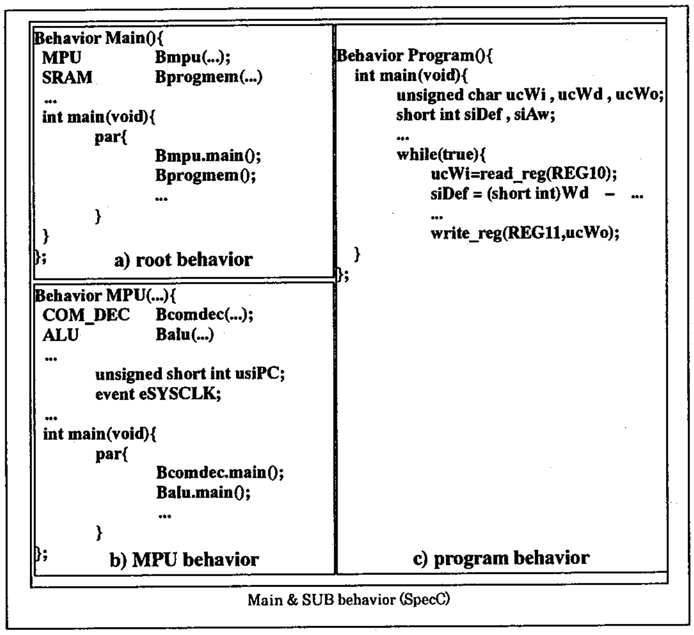


Fig. 5 Control program chart for experiment.



```
Behavior Main(){
 MPU            Bmpu(...);
 SRAM           Bprogmem(...)
 ...
 int main(void){
        par{
                Bmpu.main();
                Bprogmem();
                ...
        }
 }
};      a) root behavior
```

```
Behavior MPU(...){
 COM_DEC        Bcomdec(...);
 ALU            Balu(...)
 ...
        unsigned short int usiPC;
        event eSYSCLK;
 ...
 int main(void){
        par{
                Bcomdec.main();
                Balu.main();
                ...
        }
};      b) MPU behavior
```

```
Behavior Program(){
  int main(void){
        unsigned char ucWi , ucWd , ucWo;
        short int siDef , siAw;
        ...
        while(true){
            ucWi=read_reg(REG10);
            siDef = (short int)Wd  -  ...
            ...
            write_reg(REG11,ucWo);
  }
};
        c) program behavior
```

Fig. 6 Description examples of behavior.

A code for a system simulator is implemented according to the specifications of the hardware, the software and the MPU. Figure 7 illustrates a flow of generation and execution for the simulator from the codes.

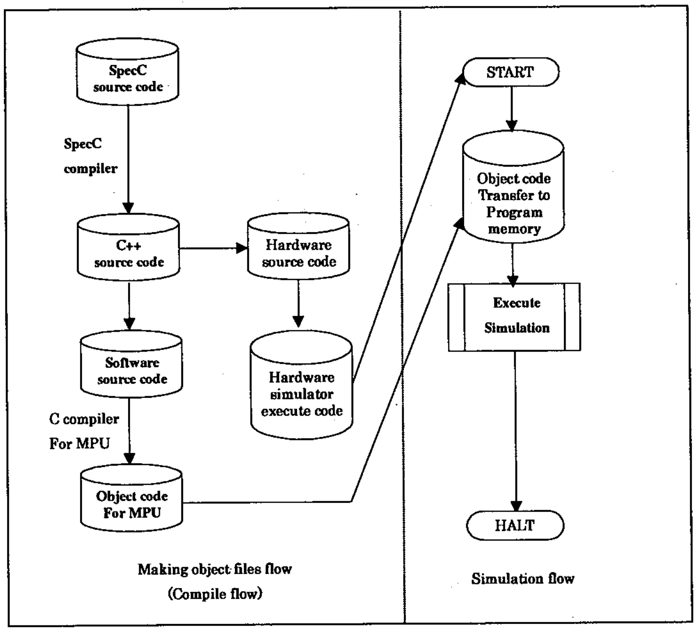
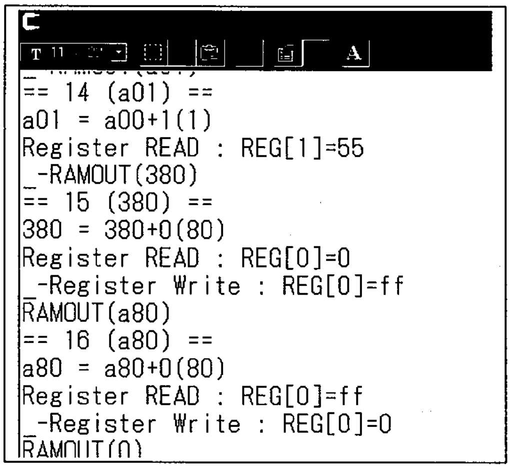


Fig. 7 Simulation executing flow.

All of the modules are written for "SpecC reference compiler V1.2". The compiler generates a "C++ source code" from "SpecC source code". Next, "g++ (GNU C++ compiler)" generates an executable simulator code from the "C++ code". The simulator starts from the main function of a main behavior. Usually "g++" compiler compiles the C++ code to generate a simulation object code. However, we cut off the codes that are parts of the program for the MPU. The codes are generated from the description part Fig.6 (c). The codes for the program are compiled to generate object codes for the MPU. The other codes are compiled to generate simulator codes. The simulator loads an object code for the MPU, and executes the code. To build a programming file for an FPGA, the object codes for the MPU merge these files. The two codes are independent each other and there are no problems for compiling and executing. At the "SpecC", a description of a behavior is closed, and it is easy to make software a capsule.

The simulator behaves hardware operations, and executes the software code by the MPU in the simulator. The hardware has some independence modules and the modules are executed independently on the simulator. The modules look like behavior level design styles. However, they are described with RTL, and act synchronously and concurrently. The synchronous signals and data paths go through the root behavior that means the top-level description of the behaviors. When the modules are added or changed, the description of the root behavior will be modified about controls for the modules.

Figure 8 illustrates simulator outputs on the application system. The outputs have following messages; "program memory output", "Executing program counter register value", "Executing code", "Selected register No. and the register's value". Figure 9 shows the simulation results for an experimental application program.



```
== 14 (a01) ==
a01 = a00+1(1)
Register READ : REG[1]=55
_-RAMOUT(380)
== 15 (380) ==
380 = 380+0(80)
Register READ : REG[0]=0
_-Register Write : REG[0]=ff
RAMOUT(a80)
== 16 (a80) ==
a80 = a80+0(80)
Register READ : REG[0]=ff
_-Register Write : REG[0]=0
RAMOUT(0)
```

Fig. 8 Executing the Simulator with SpecC.

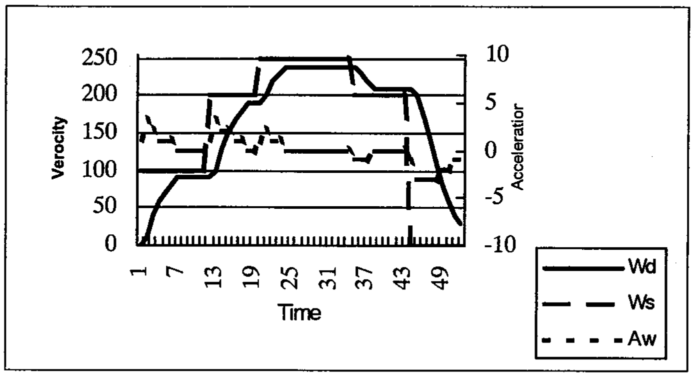


Fig. 9 Simulation output chart.

3.2 Differences of Description about Modifying MPU on "SpecC" and "VHDL"

We compare the styles about the description of the MPU on the "SpecC" and "VHDL". Each MPU has the same functions.

Before the experiments, differences of the language are expressed. VHDL description of hardware system with software is very hard, because VHDL has no function and no structure to describe the software. Accordingly, to describe the software by VHDL is too difficult and it is not good way. At the following explanation, we do not compare the descriptive style for software, and compare only re-configurable MPU description.

To evaluate the descriptive styles, we compare our proposed Spec-C-MPU with VHDL-MPU. VHDL-MPU is composed of some component blocks described by VHDL. On our proposed Spec-C-MPU, all control signals and data path go through a root behavior for re-configurable MPU. All architectures are described through root architecture of VHDL as well. To evaluate size of an effort, an ALU is supplied for each MPU. Table 1 shows the differences of the MPUs. For this addition, the codes of SpecC increase about 200Bytes. The codes of VHDL increase about 400Bytes as well. Further, the number of parts is needed to change are 3 parts for SpecC, and 8 parts for VHDL. In Spec-C-MPU, adding functions can change main behavior. The description of SpecC is abstractly and we can change smaller than VHDL.

Table 1: Increase of description for adding ALU.

	SpecC- MPU	VHDL-MPU	Note
Line	257	152	All line number of files.
Size	+200byte	+400byte	To add ALU
Change	3	8	Points. (Not Line)

3.3 Differences of Implementation for Modifying MPU ‘s Data Bus Width on an FPGA

At first, an FPGA should be decided. In this experiment, EP1K10 that is ACEX1K series distributed by ALTERA and a design tool that is “MAX+Plus2” distributed by ALTERA are selected to implement and evaluate the MPU on an FPGA. The one of ACEX1K’s merits is to have logic-cells and memory-cells independently. Best implementation can be realized by applying the merits. Figure 10 shows the map of the basic design MPU in the FPGA (EP1K10). The MPU has 16bits fixed command length and 8bits data length.

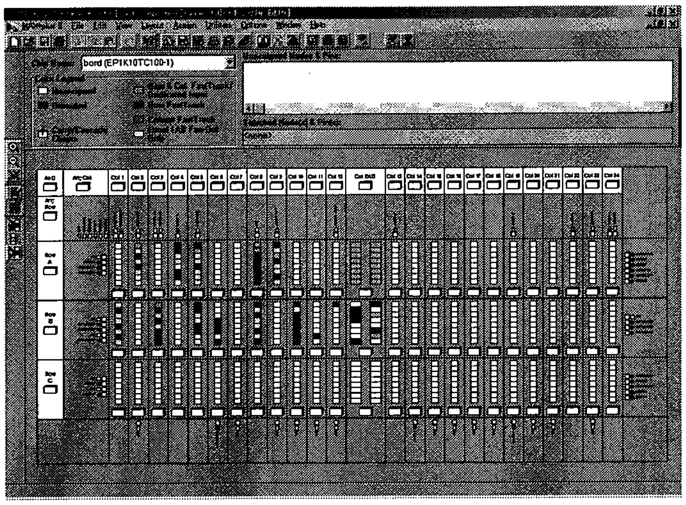


Fig. 10 Implement Basic MPU on FPGA (EP1K10).

Figure 11 shows a timing chart by the simulator. There are inner nodes for program-code-bus, but there is no input node, because the program codes for the MPU are stored in the memory-cell on the FPGA. The program codes for the MPU are included in a POF (programmer object file). The POF configures memory-cells and logic-cells of the FPGA in a same time.

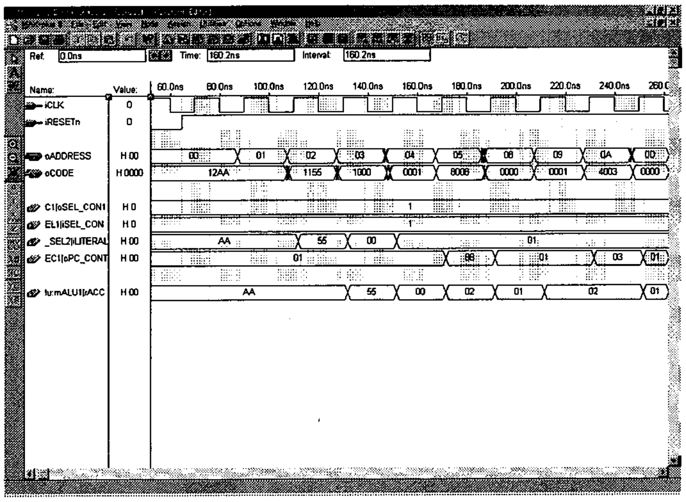


Fig. 11 Simulator results on MAX+Plus2 (EP1K10).

Next, The MPU is modified to evaluate differences of the consumption on logic-cells and memory-cells. The modifications are reductions / expansions of a data-bus in the MPU; from 8bits to 4bits, 16bits and 32bits. Furthermore a MPU without using the memory-cells is designed. A number of registers in the MPU is reduced 1/16, because a consumption of logic-cells for 256 registers is too large to build. Table 2 shows consumptions of logic-cells and memory-cells for these MPUs. The experiments show that consumption of logic-elements and memory-bits increases linearly. However increasing consumption for memory-bits is quickly. Therefore the register-file of the MPU enlarges for expanding the data bus. On a same number of the registers, the consumption of memory-bits increases as a time as an expansion time by the data bus width. The consumption of logic-elements is increase slower than the consumption of memory-bits. To expand the data bus width does not complicate the control-logics. It needs data-selectors and latch-logic. The control-logic is connected to data-selectors and latch-logics.

Furthermore, the experiments show that the memory registers built by logic-cells are not efficient to use the FPGA resource. By applying the logic-cells to register files, the consumptions of memory-cells are reduced. However, the register files built by logic-cells consume 4 times as much as control logics. By fitting the memory-cells for building register files, a memory-cell-block can have only one register file, because the memory-cell-block has only one port connecting to other logic-cells. In the experiments, the MPU with 4bit data bus width consumes same number of memory-cell-blocks for a MPU with 8bit data bus width.

Table 2: Increase in consumption cells on FPGA (EP1K10).

Data Bus Width	Memory Bits (bits)	Logic Elements
8(Logic Register)	4,096	330
4	5,120	71
8	6,144	83
16	8,192	92
32	12,288	108
EP1K10(Max)	12,288	576

3.4 Differences of Implementation for Multi MPU on an FPGA

To evaluate differences for consumptions by the number of the MPUs, some MPUs are built in an FPGA. EP1K30 is selected for next experiments, because EP1K10 does not have much memory-cells and logic-cells. Table 3 shows the result about the experiments. The results show that the MPUs consume the same volume of the logic-cells and the memory-cells for each MPU. However, our MPU is very compact and suitable to the FPGA, and can be built 3 MPUs in the FPGA at least. In this design, there is a limit of the number of MPUs by consuming memory-cells.

These experiments show how to build the re-configurable MPU in the FPGA, and problems in using the memory-cells. To realize an expansion of bus width, a number of register-cells should be reduced. Therefore, to implement some MPUs in an FPGA, to share the memory-cells between each MPU is important.

Table 3: Increase of consumption cells on FPGA (EP1K30).

Processors	Memory Bits (bits)	Logic Elements
1(Logic Register)	4,096	330
1	6,144	83
2	12,288	165
3	18,432	319
EP1K30(Max)	24,576	1,728

4.Conclusions

We have shown and described how to compose a MPU with module structure. In chapter3.1, we have shown the way to build a system applied our propose design method. In chapter3.2, it has been shown that the change and addition of each module is easily executed.

Furthermore, we have shown the way to add an ALU and the efficiency of our propose design method. In chapter3.3, we have shown the differences of descriptions and the consumptions of the FPGA resources by changing the bus width on each MPU. Furthermore, in chapter3.4, we have shown how to build some processors and the problems in an FPGA.

We have confirmed that we can realize a user demand in consideration of the trade-off. The trade-off is following that. To modify data-bus-width, we must pay attention to consumptions of memory-cells by a register-file. To add the MPU, we must pay attention to consumptions of memory-cells and logic-cells. Furthermore, for efficient design, we must realize a function to share the register-files among MPUs.

We have expressed that our proposed MPU and its design technique are effective and useful to find a trade-off expanding MPUs in an FPGA.

As the future work, we will study about the following extensions.

- To build a multi processing system in an FPGA.
- To share the memory-cells on multiprocessor systems in an FPGA.
- To share the function modules between processors in an FPGA.
- To build a support functions for an embedded system.

By realizing extensions, our proposed processor system will obtain excellent architectures to build an embedded system. The excellent architectures will realize following systems.

- Real-time system without Real-time Operation System
- Multi processor system with register sharing
- Multi processor system with program memory sharing

We consider that these systems are very efficiency to built compact and flexible embedded systems.

References

[1] M. Gschwind, V. Salapura, D. Maurer, "FPGA Prototyping of RISC Processor Core for Embedded Applications" in IEEE Trans. VLSI-Systems, Vol.9, No.2, pp.241-pp.250, 2001.

[2] B. K. Tan, T. Ogawa, R. Yoshimura, K. Taniguchi, "A Reconfigurable Digital Signal Processor" in IEICE Trans. Electron, Vol.E81-C, No.9, pp.1424-pp.1430, 1998.

[3] S. Honda, H. Takada, "Evaluation of Applying SpecC to the Integrated Design Method of a Device Driver and a Device" in IPSJ Trans. Vol.43, No.5, pp.1214-pp.1224, 2002.

[4] D. Gajski, J. Zhu, R. Domer, A. Gerstlauer, and S. Zhao, Spec C: Specification Language and Methodlogy, Kluer Academic Publishers, 2000.