A Modeling of a Dynamically Reconfigurable Processor using SystemC

Junji Kitamichi† Koji Ueda‡ Kenichi Kuroda†

† School of Computer Science and Engineering,
The University of Aizu,
Tsuruga, Ikki-machi, Aizu-Wakamatsu,
Fukushima 965-8580, JAPAN
{kitamiti,kuroda}@u-aizu.ac.jp

‡ NEC Electronics Corporation
1753, Shimonumabe,
Nakahara-Ku, Kawasaki,
Kanagawa 211-8668, JAPAN

Abstract

Recently, Dynamically Reconfigurable Processors (DRPs) have been proposed. In this paper, we describe a model of a DRP using a Dynamic Module Library (DML), which we have developed for the modeling of general-purpose dynamically reconfigurable systems. The DML is an extended SystemC library and enables the modeling of the dynamic generation and elimination of modules, ports and channels and the dynamic connection and dispatch between port and channel. Using the DML, we can model the DRP naturally. The architecture of the proposed DRP is based on an MIPS-type architecture and supports the instructions, which are for the dynamically reconfigurable operational units and for their generation and elimination. We describe the proposed DRP model and its evaluation results.

1 Introduction

Recently, many types of Dynamically Reconfigurable Processors (DRPs) based on FPGA technology have been proposed[1] [2] [3]. DRPs are implemented on a unique Dynamically Reconfigurable Architecture (DRA), and a specialized design environment is provided for the DRP. In the case of the system design for a new application-specific Dynamically Reconfigurable System (DRS), the existing design methods and CAD systems can not deal with this system design. In the case of a new DRP architecture design, a design environment that is independent of the specific DRA will be needed at the system level, which is the beginning of system design.

The system level design[4] is more abstract that the conventional Register Transfer (RT) level design, and system specification languages, such as SystemC [5], SpecC[6], and SystemVerilog[7], are proposed.

We have been developing a Dynamic Module Library (DML) [8] for the modeling of general DRSs from the system level to the RT level, which provides the dynamic generation and elimination of a module and the dynamic connection and dispatch of the port and channel. In SystemC2.1.v1, a dynamic process is introduced, which enables the dynamic process generation and elimination. However, using this process, the modules, ports, and channels cannot be generated and eliminated dynamically and cannot be connected and dispatched dynamically after the simulation has started. Using the DML, the dynamic generation and elimination of a module and the dynamic connection and dispatch of the port and channel are implemented and modeled naturally.

In this paper, we describe a DRP model at the RT level using the DML and its evaluation results. The proposed processor is based on an MIPS-type processor which supports the dynamically reconfigurable operational units that are dynamically generated and eliminated as well as the instructions for dynamic reconfiguration and their operations.

2 Background

2.1 System Modeling using SystemC

SystemC[5] is a system description language. SystemC has the ability for system modeling using a class library of C++. System models from the abstraction level to the concrete RT level can be described in SystemC, and they can be simulated at the different levels.

The behavior of the system is described in the modules. In the module, the definitions of input and output ports and the definitions and initializations of processes, such as the behavior of the module, the generations of instances of sub-modules, the connection with ports in the sub-modules

1063-9667/08 $25.00 © 2008 IEEE
DOI 10.1109/VLSI.2008.13

to channels, and the behavior of processes are described. The modules are defined by inheriting the *sc_module* class, a class of SystemC. The channels are the communication paths between modules, and the ports are connected to channels and the port of modules. For example, *sc_in* and *sc_fifo* are types of an input port and a FIFO channel respectively.

In the older version of SystemC than 2. 1.v1, processes are initialized only at the start of a simulation and are never generated or eliminated during a simulation. The idea of a dynamic process is added in SystemC 2. 1.v1. With the dynamic process, it is possible to activate processes after the simulation has started. However, each port and channel is defined as static and is never generated or eliminated dynamically during the simulation in SystemC 2. 1.v1.

2.2 Dynamically Reconfigurable Architectures

Recently, Dynamically Reconfigurable Architectures (DRAs)[9] have been proposed. A DRA is based on FPGA, whose configuration memory is SRAM, and can be reconfigured while the system is operating. There are many types of DRAs[1][2][10][11], such as the multi-context type / partially reconfigurable type and the fine-grain / course-grain types. The reconfigurable units of fine-grain architectures are constructed by Look-Up Tables (LUTs) that use memory technology. LUTs can be constructed as any logic function, so every region can be efficiently used. On the other hand, ALUs construct reconfigurable units of coarse-grain architectures; therefore, they can compute numerical operations faster than fine-grain architectures. However, some reconfigurable units cannot be utilized, and the usage of the area is worse.

Multi-context DRA[1] and partially DRA[10] [11] make up one classification of DRAs. Multi-context DRAs have some configuration data memories, and the circuits are reconfigured by switching the configuration data. The switching of the configuration data takes place at once. The contents of unused configuration memories can be rewritten. In the partially DRA, the reconfiguration of circuits corresponds to the rewriting of the configuration data in some areas.

2.3 Dynamic Module Library

We describe a modeling method of the systems that includes the dynamic generation and elimination of modules and interface at the system level. In the DRS, modules and an interface are generated or eliminated while the system is running. In order to express these behaviors naturally at the system level, modules, ports and channels have to be generated or eliminated dynamically. However, their generation and elimination cannot be performed after the simulation has started, even if SystemC 2.1.v1 is used. We have proposed a Dynamic Module Library(DML) [8] including dynamic modules, dynamic ports, and dynamic channels to solve these problems.

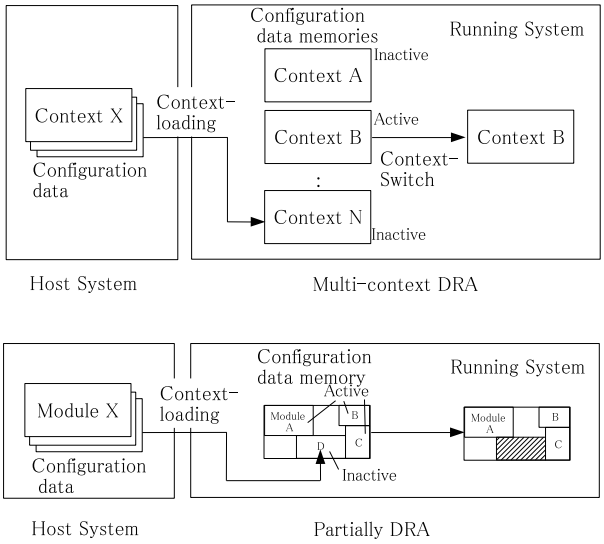

Figure 1. Outline of two types of DRAs

If the DML is used in modeling, the dynamic module class *DynamicModule*, which collects functions, such as *newmod()*, *new* and *delmod()*, for the dynamic module, port and channel is used rather than the *sc_module* class. In a dynamic module, the usual ports, such as *sc_port* and channels, such as *sc_fifo* cannot be used for their definition. In addition, the *sc_port* and *sc_fifo* have a binding method to connect with each other, but they do not have any method to dispatch. The dynamic port, such as *dc_port* and dynamic channel, such as *dc_fifo* are used in a dynamic module and have methods to connect and dispatch.

The DML has a channel pool, which is a set of static channels, and manages the connection and dispatch of the channel and port in the channel pool. The dynamic ports are implemented and managed in the same way as the dynamic channels.

The dynamic module, ports and channel are implemented by the specific hardware resources on the adopted DRA. In the case of Multi-context DRA, one or more dynamic modules may be mapped into the context data, and dynamic channel may be mapped into the store operation into the memory outside DRA and the reload operation from the outside memory.

If the dynamic modules were used in the design, the simulation time could be increased because of the overhead of the operations of the DML. Therefore, several acceleration methods[8] are built in the DML.

Some modeling methods for DRSs using SystemC have

been proposed. The method proposed in [12] has some constraints on the modeling, such that all modules must be on same level of hierarchy and instantiated in th same component. The target modeling class in [13] and [14] is more limited type of DRAs than our proposed method.

3 Proposed Processor Architecture

We describe a proposed DRP architecture. The proposed processor has some operational units, which are generated and eliminated dynamically. It consists of a processor core and an Extended Functional Unit(EFU), which contains some Dynamically Reconfigurable Operational Units(DROUs). The overview of the proposed processor is shown in Figure 2. In this paper, we deal with the modeling independently of a specific DRA. In the modeling, we use SystemC and the DML.

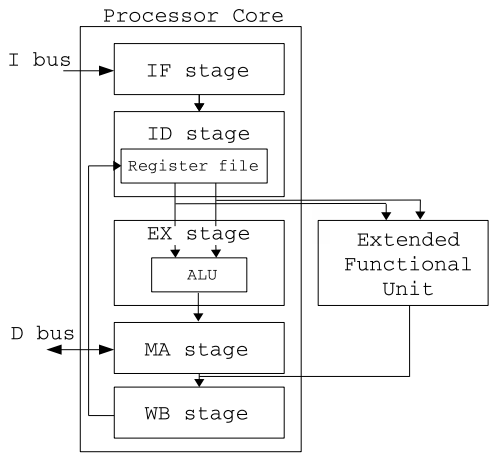

Figure 2. Block Diagram of the Proposed Processor

3.1 Processor Core

The proposed processor is a 32-bit pipelined processor that executes the subset of instructions of MIPS[15] in the 5-stage pipeline. The instruction set consists of about 50 instructions, such as integer/logical operations, Load/Store, and Jump/Branch, and each instruction is executed in order. The processor has an integer- and a floating-point register files, and has the data transfer instructions between each other.

In the execution of instructions, the proposed processor executes the Instruction Fetch(IF), the Instruction Decode(ID), the EXecution(EX), the Memory Access(MA) and the Write Back(WB) stages in order. In some instructions, the processor skips some stages. For example, in the case of integer arithmetic instructions, it skips the MA stage and forwards the result to the ID stage or the EX stage. The processor has an integer ALU in the EX stage as the static operational unit, and executes the Load/Store instructions, Addition/Subtraction/logical instructions, and Multiplication/Division instructions at 5, 4 and about 35 clocks.

The instructions on the dynamic reconfiguration are the Dynamically Generation (DG) instructions of the DROU and the OPerational (OP) instructions using the DROU. The DROUs are generated in the EFU, execute the operation for the data received from the ID stage, and send the results to the WB stage.

A PRISC[16] is developed with the DRP closely connected with the RF, as in our proposed processor.

3.2 EFU and its Interface to the Processor Core

The EFU has two input and one output ports and contains several DROUs. The EFU contains the values of the Configuration Address (CA) and Block Address (BA) in the table in order to manage the status of the reconfiguration.

The DG and OP instructions are also 32-bit width. The DG instructions contain a 6-bit CA field and a 3-bit BA field. The value of the CA field denotes the type of DROUs, and the value of the BA field denotes the location of the dynamic reconfiguration blocks in the EFU.

We can describe more abstract specification of the construction of instructions and the type of operations using the parameters and abstract data types. In the proposed model, for the simplicity, we adopted the fixed construction of instructions and the fixed types of operations.

We assume that the number of clocks for the generation, elimination of the dynamic reconfiguration blocks and the execution of the operation is defined according to the types of DROUs and can be given as the parameter. The ID stage is stalled during the cycles specified by these parameters.

The OP instructions consist of 2 or 3 operands. A 3-bit address field, which is one of the operands, specifies the location of dynamic reconfiguration blocks. The operational results are sent to the WB stage.

The execution of these instructions is shown in Figure 3. In the case of the DG instructions, the ID stage sends a signal *Config_en*, which indicates the start of dynamic reconfiguration, and a signal *Config_adr*, which indicates the type of operational units to be dynamically reconfigured, to the EFU. The EFU spends the time *Creating_time* when the through DROU must be reconfigured dynamically. If the DROU has been configured and this DROU has to be eliminated for new unit, the EFU spends the necessary time *Deleting_time* and then issues the signal *Genend* to the ID stage.

In the case of the OP instructions, the ID stage sends source operands *FP_data* or *INT_data*, i.e., floating point data or integer data, respectively, a signal *Drou_op*, which is

the type of operations, and and a signal *Drou_adr*, which is the location of dynamic reconfiguration blocks at the EFU. The execution time at the DROU is specified as the parameter *Running_time*. The EFU sends the execution results to the WB stage and sends a signal *Exend* to the ID stage. By receiving the signal *Exend*, the ID stage restarts the issue of instructions.

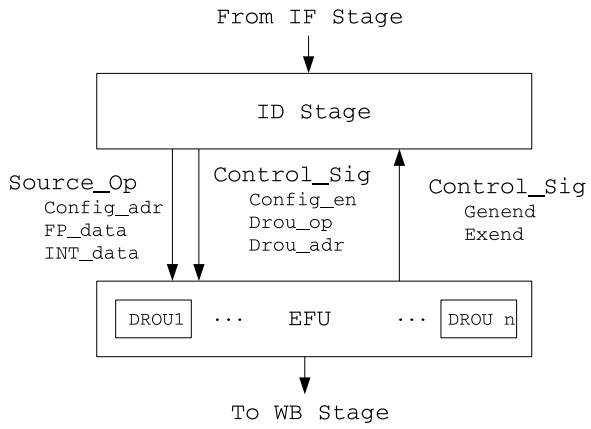

Figure 3. Interface of EFU and Processor Core

3.3 Modeling of the Proposed DRP

We describe the proposed processor model using the DML. We describe the IF, ID, MA, and WB stages as the static modules in the original description style of SystemC at the RT level. Then we omit these modeling. These are described at the RT level, Hereafter, we describe the description about the EX stage, which concerns the dynamically reconfiguration.

(a)Dynamically Reconfigurable Execution Unit

We modeled 4 types of modules; addition, subtraction, multiplication, and division for the floating point as the DROUs. The numbers of clocks for the generation and elimination of modules and operational clocks are given as parameters.

The part of the description of a multiplication DROU is shown in Figure 4. The *Dynamic_Mulf* class is defined by inheriting the *DynamicModule* class. This class has the dynamic ports *dc_in* and *dc_out* and user processes, such as *running*.

The *running* process executes the floating point multiplication using the values from the input ports *fsin* and *ftim* and sends the results through the output port *fout*. The behavior of the *running* process, which is executed through the internal variables *fs_signTmp*,*ft_signTmp* and etc., is described in the original description style of SystemC. *mulf_etime* is used as the parameter, which is the period of the operation, and the *running* process outputs the result *sign1_tmp*, *exp5_tmp*, and *man5_tmp* through the port *fout* and waits for the next operation . The description of the multiplication process is omitted in Figure 4.

```
#include "DynamicModule.h"

class Dynamic_Mulf: public DynamicModule
{
public:
  dc_in<sc_uint<DWORD> > fsin,ftin;
  dc_out<sc_uint<DWORD> > fout;
  dc_in<bool> dc_en;

  SC_HAS_PROCESS(Dynamic_Mulf);
  Dynamic_Mulf(const char *name, int ctime,
               int dtime, sc_time_unit tunit):
    DynamicModule(name, ctime, dtime, tunit)
  {

  }
  ~Dynamic_Mulf(){};

  void running();
  void creating();
  void deleting();
};

void Dynamic_Mulf::running(){
       :
  while(true){
    while(dc_en.read()==false){
      if( dwait(dc_en.value_changed_event()) )return;
    }

    fs_signTmp = fsin.read()[31];
    ft_signTmp = ftin.read()[31];
    fs_expTmp =  fsin.read().range(30,23);
    ft_expTmp =  ftin.read().range(30,23);
    fs_manTmp =  fsin.read().range(22,0);
    ft_manTmp =  ftin.read().range(22,0);

       :

    dwait(mulf_etime, SC_NS);
    fout.write( (sign1_tmp, exp5_tmp, man5_tmp) );
    dwait(mulf_waits, SC_NS);
  }
}
```

Figure 4. Description of DROU

(b)Control of Dynamic Reconfiguration in EFU

We describe the model of dynamic reconfiguration control in the EFU. Part of the description of the model is shown in Figure 5. This control unit has 4 states;*Idle*, *Creating*, *Running*, and *Deleting*. **(1)**The control unit transits the *Idle* state by the *Reset* signal. **(2)**It transits the *Creating* state by the *Config_en* signal from the ID stage. In this state, it stalls for the *Creating_time*. **(3)**Then it sends the *Genend* signal to the ID stage and transits the *Running* state. **(4)** and**(5)**In the *Running* state, the OP instruction is executed. It stalls for the *Running_time*, sends the executed results to the WB stage, sends the *Exend* signal to the ID stage, and waits for the next instruction issue. **(6)**By the arrival of the *Config_en* signal, it transits the *Deleting* state and stalls for the *Delet-*

ing_time. **(7)**To reconfigure a new DROU, it transits the *Creating* state.

Creating_time and *Deleting_time* are the parameters, which are defined according to the scale of module, adopted DRA, and etc.

The transitions in **(6)** and **(7)** are the behavior in the case that the DROU has been configured and this operational unit should be eliminated for the new unit.

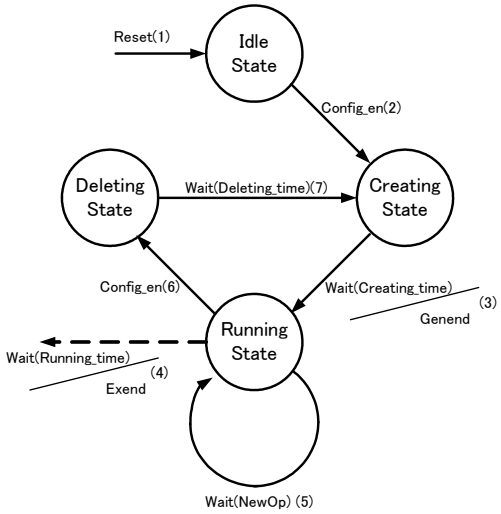

Figure 5. State Diagram in EFU

We describe the behavior for the dynamic reconfiguration. This description, which is the part of step **(6)** reported above, is shown in Figure 6. The behavior of the reconfiguration for a new DROU *Dynamic_Addf* at the *unit[i]*, which denotes the specified block in the EFU by the issued instruction is described. The controller executes following procedures.

(1) With the method *detach*, the channels *dina*, *dinb*, and *dout* and the ports *fsin*, *stim*, and *fout* are dispatched, respectively. **(2)**With the method *delmod*, the module is eliminated, and the time to eliminate the module is passed. **(3)**With the method *new*, a new DROU is generated. **(4)**With the method *bind*, the channels *dina*, *dinb*, and *dout* and new generated ports are connected, respectively. **(5)**The time *addf_ctimeto* to generate the module is passed.

Other channels *dina*, *dinb*, and *dout* are static and connected to the outer ports of the EFU.

4 Experimental Results

We show the experimental results of the proposed processor model, which is described using SystemC 2.1.v1 and the DML. The experimental environment is a Xeon 3.2GHz, Mem 4GB, Linux 2.6.9, GCC 3.4.6, and SystemC 2.1.v1 by OSCI.

```
// If unit[i] is used and reconfigured to Dynamic_Addf
          unit[i]->fsin.detach(dina);
          unit[i]->ftin.detach(dinb);
          unit[i]->fout.detach(dout);
          unit[i]->dc_en.detach(ex_en);
          unit[i]->delmod();
          wait(unit[i] -> dtime, SC_NS);
          unit[i] = new Dynamic_Addf("addf",
                        addf_ctime, addf_dtime, SC_NS);
          unit[i]->fsin.bind(dina);
          unit[i]->ftin.bind(dinb);
          unit[i]->fout.bind(dout);
          unit[i]->dc_en.bind(ex_en);
          unit[i]-> dtime = addf_dtime;
          wait(addf_ctime, SC_NS);
         }
                              :

//Connection Input Port and Channel
void Dynamic_Control::data_in() {
  while(true) {
    wait();
    dina = in0.read();
    dinb = in1.read();
  }
}

//Connection Channel and Output Port
void Dynamic_Control::data_out(){
  while(true) {
    wait();
    if(exend_tmp==true){
      out.write(dout.read());
    }
  }
}
```

Figure 6. Description of the Control for DROUs

The Mux model, which is modeled in the original description style of SystemC, is prepared as a comparison with the proposed processor using the DML. In the Mux model, the generation and elimination of a module are described using some multiplexers and de-multiplexers.

The lengths of the codes after eliminating comments, blank lines, and macros for C++ with the preprocessor of *gcc* are shown in Table 1.

Table 1. Code lengths of the models

	S Modules	D Modules	Control
Mux model	8248	787	507
Proposed method	8248	751	588

In Table 1, "S Modules" denote the static modules of data path and control units and the common description in the proposed model and the Mux model. "D Modules" denote the descriptions of DROUs in the proposed model and the descriptions of static DROUs, multiplexers, and de-multiplexers in the Mux model. "Control" denotes the description of the control of the dynamic reconfiguration. In the proposed model, this includes the descriptions of the generation and elimination of modules, ports, and channels

and the connection and dispatch of ports and channels.

In the "D Modules," the description of the Mux model is slightly longer, and, in the "Control," that of the proposed model is longer. The difference in the total descriptions is slight.

In the simulation, the times of 1,000 executions of DG and OP instructions are measured. As a result, the time of the Mul model is 6.13 sec., that of the proposed method without the acceleration method is 6.90 sec., and that of the proposed method with acceleration methods is 6.72 sec. The Mux model is the fastest, although the difference among them is slight. In the case of a low frequency of reconfiguration, faster simulation of the proposed method is possible[8].

The memory usage of the DML depends on the number of dynamic module types, and is fixed during the simulation. In the experiment, it is dominated mainly by the user processes of the static modules, especially in the ID stage.

5 Conclusion

We described a DRP model which can dynamically generate and eliminate the operational units using the DML. We can easily and naturally model the generation and elimination of the DROUs in the proposed processor.

The proposed model can be described as having a similar size as that of the Mux model, which uses multiplexers and de-multiplexers for switching the function, and similar speed simulation is possible. We showed the variability of DML for the modeling of general-purpose DRPs.

In the proposed model, the processor core is modeled using SystemC at the near-RT level, and, on the contrary, the parts of the dynamic reconfiguration are modeled at the abstracted level. We will refine the description level of the total system from the system level to the RT level and apply our method for other DRP types. We research the system partitioning from system model into static devices and dynamically reconfigurable devices and the technology mapping of dynamic modules and channels to a concrete DRA.

Acknowledgment

This research has been supported by the Kayamori Foundation of Information Science Advancement.

References

[1] H. Nakano, T. Shindo, T. Kazami, and M. Motomura, "Development of Dynamically Reconfigurable Processor Lsi," NEC Technical Journal,Vol.56,No.4,pp.99-102, 2003.

[2] T.Toyo, H.Watanabe, and K.Shiba. "Implementation of Dynamically Reconfigurable Processor DAP/DNA-2," Proc, of IEEE VLSI-TSA Int. Sympo. on VLSI Design, Automation & TEST, pp.321-322, Apr.2005.

[3] J.E.Carrillo and P.Chow, "The Effect of Reconfigurable Units in Superscalar Processors," Proc. of the 2001 ACM/SIGDA 9th Int. Sympo. on Field Programmable Gate Arrays, pp.141-150, Feb.2001.

[4] R. Bergamaschi and W. Lee, "Designing Systems-on-Chip Using Cores," Proc. of the 37th Conf. on Design Automation, pp.425-430, Jun.2000.

[5] Open SystemC Initiative, "SystemC 2.1 Language Reference Manual," http://www.systemc.org.

[6] SpecC Technology Open Consortium, http://www.specc.org.

[7] Accellera Organization, Inc., http://www. systemverilog.org.

[8] K. Asano, J. Kitamichi, and K. Kuroda, "Proposal of Dynamic Module Library for System Level Modeling and Simulation of Dynamically Reconfigurable Systems,"Proc. of 20th Int. Conf. on VLSI DESIGN(VLSI DESIGN 2007),pp. 373-378, 2007.

[9] M. Gokhale and P. S. Graham," Reconfigurable computing : Accelerating computation with field-programmable gate arrays," SPRINGER,2005.

[10] Xilinx,"Vertex-4 Configuration Guide,"http://direct.xilinx.com/bvdocs/userguides/j_ug071.pdf, UG071 (v1.3), 2005.

[11] T. Shiozawa, N. Imlig, K. Nagami, K. Oguri, A. Nagoya, and H. Nakada, "An Implementation of Longest Prefix Matching for IP Router on Plastic Cell Architecture," Proc. of FPL 2000, LNCS 1896, pp.805-809, 2000.

[12] A. Pelkonen, K. Masselos, and M Cupák, "System-Level Modeling of Dynamically Reconfigurable Hardware with SystemC,"Proc. of Int. Paralles and Distributed Processing Symp.(IPDPS'03),pp.174-181,2003.

[13] A. V. de Brito, E. U. K. Melcher, and W. Rosas, "An open-source tool for simulation of partially reconfigurable systems using SystemC,"Proc. of 2006 Emerging VLSI Technologies and Archtectures(ISVLSI'06), pp.434-435,2006.

[14] P. A. Hartmann, A. Schallenberg, F. Oppenhaimer, and W. Nebel, "OSSR+R:Simulation and Synthesis of Self-adaptive Systems,"Proc. of Field Programmable Logic and Applications(FPL2006),pp.177-182, 2006.

[15] G. Kane, "MIPS RISC Architecture-R2000/R3000-," Prentice-Hall, 1988.

[16] R.Razdan and M.Smith, "A High-Performance Microarchitecture with Hardware-Programmable Functional Units," Proc. of 27th the Annual Int. Sympo. on Microarchitecture,pp.172-180, Nov.1994.