設 計 自 動 化 79－4
（1 9 9 6. 2. 9）

# 異なる抽象化レベルを持つシミュレーションモデルの生成 － レジスタ転送の関数表現に基づく生成法 －

内田 健　　貴家 仁志　　山田 昭彦

東京都立大学 工学部 電子・情報工学科

〒192-03 東京都八王子市南大沢 1-1
Tel: 0426-77-2745 Fax: 0426-77-2756
E-mail: uchida@isys.eei.metro-u.ac.jp

あらまし　プロセッサのシミュレーションにおいて重要なことは，シミュレーションの目的に適したシミュレーション速度とシミュレーション精度を選択することである．本報告では，異なる抽象化レベルを持つシミュレーションモデルの生成手法を提案し，その生成アルゴリズムを与える．さらに，レジスタ転送を関数表現することにより，これらの生成手法が関数合成によって実現されることを示す．最後に，本手法を DLX プロセッサに適用した結果，シミュレーションモデルの生成に約 30 秒かかることを確認した．

キーワード　シミュレーション，マイクロプロセッサ，トレーステーブル，抽象化手法

# Generating a Hierarchical Simulation Model on the Basis of Functional Model of Register Transfers

Takeshi UCHIDA　　Hitoshi KIYA　　Akihiko YAMADA

Department of Electronics and Information Engineering, Faculty of Technology,
Tokyo Metropolitan University

1-1 Minami-Ohsawa, Hachioji-shi, Tokyo, 192-03
Tel: 0426-77-2745 Fax: 0426-77-2756
E-mail: uchida@isys.eei.metro-u.ac.jp

**Abstract** In simulation of processors, it is important to select suitably both simulation speed and precision for the purpose of simulation. In this paper, we propose two abstraction methods for generating each simulation model at different levels of abstraction; then provide its generation algorithm. Furthermore, we show these methods are implemented as the function composition based on the functional representation of register transfers. Finally, applying these methods to DLX processor, the simulation model is generated in less than 30 sec. for 30 instructions.

Key words *Simulation, Microprocessor, Trace Table, Abstraction Methods*

## 1 Introduction

Simulation of processors is one of the indispensable techniques for evaluating performance, numerical analysis of application programs, exploiting parallelism, and so on [6]–[10]. Although there exist in general tradeoffs between simulation speed and simulation precision, these two factors are suitably selected for the purpose of simulation.

On the other hand, retargetable simulation models are required in hardware/software codesign, because the reconstruction of simulation models is required when the architecture of a target processor is changed [5][9].

For these reasons, various techniques for the simulation have been investigated. However, these techniques are almost restricted to the case of a fixed target processor or the case of fixed precision [9][10][13].

In order to provide the simulation model that is retargetable and can simulate a target processor at different levels of abstraction, we therefore propose abstraction methods to generate this model automatically.

## 2 Basic concepts

In this section, we summarize our hierarchical simulation model (HSMAP), which can simulate state transitions following execution of instructions in a target processor, and its generation flow.

HSMAP has a hierarchical structure of abstraction. This structure is similar to that of the interpreter model [1]. In this structure, HSMAP has different views of behavior and of time. For this reason, HSMAP can simulate state transitions with different simulation cycles and different behavioral granularity.

In this work, we treat processors, which use no parallelism of instructions, as target processors.

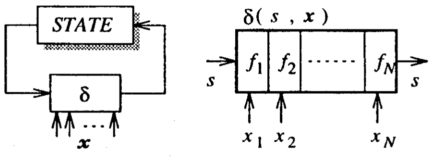
### 2.1 The hierarchical structure of abstraction

Table 1 shows the basic structure of abstraction and the different views of behavior and of time at each level of abstraction: In this paper, we call a behavioral and a temporal unit in these views a primary operation and a regular interval, respectively. These two views are important factors that affect both simulation speed and simulation precision. As a result, HSMAP can select appropriate granularity of behavior and of time for the purpose of simulation.

### 2.2 The hierarchical simulation model

As shown in the above, HSMAP has the different level of abstraction. However, we represent HSMAP as a finite state machine in a unity form (shown in Fig. 1).

In Fig. 1, *STATE* and $\delta$ indicate a state (e.g., an instruction memory, a data memory and registers) and a state transition function following execution of instructions, respectively. The input $\boldsymbol{x}$ is an integer vector that controls execution of instructions.



(a) FSM model (b) Function $\delta$

Figure 1: HSMAP

The function $\delta$ consists of some functions corresponding to primary operations executed at regular intervals (e.g., instruction cycles, machine cycles and so on) as shown in Fig. 1, because each instruction is carried out by serial execution of these operations, which are specified according to an abstraction level (shown in Table 1). For the above reason, the vector $\boldsymbol{x}$ has elements where only single element is 1 and others are 0; this 1's element selects exactly the primary operation carried out at the regular interval.

As a result, HSMAP can be defined as the following unity form:

$$s = \delta(s, \boldsymbol{x}) \tag{1}$$

$$\delta(s, \boldsymbol{x}) = \begin{cases} f_a(s), & \exists a \ x_a = 1 \ \wedge \\ & \forall n \neq a \ x_n = 0 \\ s, & \forall n \ x_n = 0 \end{cases} , \tag{2}$$

$$\boldsymbol{x} = (x_1, x_2, \cdots, x_N), \tag{3}$$

where $s$ is a state, $n$ and $a$ are integers ($1 \leq n, a \leq N$), $x_n \in \{0, 1\}$ and $f_1, \cdots, f_N$ are functions corresponding to the primary operations.

### 2.3 Generation flow

Figure 2 shows the flow to generate HSMAP. In this figure, the input, *RTL*, indicates the specification, which consists of conditional register transfers at the clock phase level with no detail to control execution of instructions. The output in this flow is *SIM_MOD*, which is defined as Eq. (1).

Table 1: Structure of abstraction hierarchy

| ABSTRACTION LEVEL | TEMPORAL VIEW | BEHAVIORAL VIEW |
|---|---|---|
| INSTRUCTION | instruction cycle | instruction |
| MICRO | machine cycle | micro-instruction |
| CLOCK | clock cycle | register transfer |
| PHASE | clock phase | register transfer |

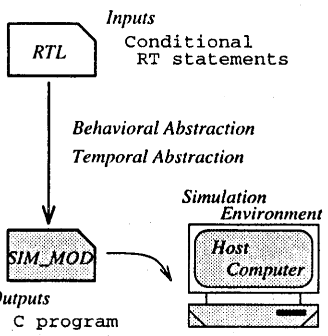


Figure 2: Generation flow of HSMAP

*RTL* is represented as the functional model described in Sect.**3**. From *RTL*, *SIM_MOD* is generated by the abstraction methods described in Sect.**4**:

- behavioral abstraction
- temporal abstraction

*SIM_MOD* is implemented as a source code of C language; this code is compiled on a host computer. As a result, we can get a simulation model that is executable on the host computer.

## 3 The functional model of register transfers

In order to represent register transfers as a functional model, we consider applying the functional model of Gannon et al. [3] to register transfers. In this section, we propose a new functional model of register transfers so that abstraction methods can be treated as operations on this functional model.

### 3.1 Notations

In the rest of this paper, the following notations are used [2] [3]:

$s$ : a state function: $VAR \rightarrow VAL$, where $VAR$ and $VAL$ are a set of variable names and of values, respectively. Especially, $s_i$ and $s_o$ are used to indicate the input state and the output state, respectively.

*state* : a set of $s$.

*expr* : an expression: $state \rightarrow VAL$.

*assign* : an assignment: $state \rightarrow state$. This has a form: $v := expr$, $v \in VAR$.

If the variable $v$ has the value $a$, then the relation between $v$ and $a$ is written as

$$[v](s) = s(v) = a, v \in VAR, a \in VAL, \qquad (4)$$

where $[\cdot]$ denotes the function that computes the same values as its argument "$\cdot$". Let $X$, $Y$ and $+(X,Y)$ be expressions, then its value in the state $s$ is given as

$$[+(X,Y)](s) \;=\; +([X](s),[Y](s)). \qquad (5)$$

For Boolean expressions, it is almost the same. Let $X$ and $Y$ be expressions; and let $X \geq Y$ be a Boolean expression, then its value in the state $s$ is given as

$$[X \geq Y](s) = \begin{cases} true, & [X](s) \geq [Y](s) \\ false, & [X](s) < [Y](s) \end{cases}. \qquad (6)$$

Let $A$ be an assignment, $v := expr$ ($v \in VAR$), then its functional representation is given as

$$\begin{aligned}[A] \;=\; & \{\,(s_i,s_o) \mid s_o = s_i \\ & \quad except\ that\ s_o(v) = s_i(expr)\,\}. \end{aligned} \qquad (7)$$

From Eq. (6) and Eq. (7), a conditional assignment, "**if** $\alpha$ **then** *assign*", is obtained as

$$\begin{aligned}[\text{if } \alpha \text{ then } A] \;=\; & \{(s_i,[A](s_i)) \mid [\alpha](s_i)\} \\ \cup\; & \{(s_i,s_i) \mid \neg[\alpha](s_i)\}, \end{aligned} \qquad (8)$$

where $\alpha$ and $A$ indicate a Boolean expression and an assignment, respectively.

### 3.2 An extended functional model

In order to applying the function model [3], we extend this model as follows: We firstly define a *conditional register transfer* as a set of assignments, which are executed concurrently on its condition:

$$\textbf{if } \alpha \textbf{ then } \{R_1, R_2, \cdots, R_M\}, \qquad (9)$$

where $R_1,\cdots,R_M$ indicate concurrent assignments and $\alpha$ is represented as a Boolean expression. To simplify, we use "$\alpha \Rightarrow r$" instead of Eq. (9).

Secondly, we consider the functional model corresponding to Eq. (9). There exists an important difference between register transfers and Pascal statements, the target of the model [3]: register transfers allow concurrent assignments. By considering this concurrence, we have the functional model of register transfers:

$$\begin{aligned}[r] &= \Big\{(s_i, s_o) \mid s_o = s_i \textit{ except that} \\ &\qquad \bigwedge_{1\le m\le M} [v_m](s_o) = [expr_m](s_i)\Big\}, \qquad (10)\end{aligned}$$

where $v_m \in VAR\ (m = 1, 2, \cdots, M)$ and $\forall i,j\ v_i \neq v_j$. This equation has the same form as Eq. (9) if $M = 1$.

Finally, from Eq. (8) and Eq. (10), we can also get the functional model of the *conditional register transfers* as the following form:

$$\begin{aligned}[\alpha \Rightarrow r] &= \{(s_i, [r](s_i)) \mid [\alpha](s_i)\} \\ &\cup\ \{(s_i, s_i) \mid \neg[\alpha](s_i)\} \qquad (11)\end{aligned}$$

## 4 Proposed Methods

HSMAP has the different views of behavior and of time as shown in Table 1. In order to generate automatically each simulation model that has both a different primary operation and a different regular interval, we consider two abstraction methods: One is behavioral abstraction to synthesize each primary operation at an upper level from primary ones at a lower one. The other is temporal abstraction to generate both each starting point of the primary operations at the upper level and the order of their execution.

### 4.1 Behavioral abstraction

A simple example of behavioral abstraction is shown in Fig. 3, where the primary behavioral operations ($M_1$, $\cdots$, $M_5$) at the micro level are synthesized from those ($C_1$, $\cdots$, $C_{11}$) at the clock level. Besides, $\pi_1$, $\cdots$, $\pi_5$ indicate sets, each of which includes primary operations carried out in the same interval at the upper level (micro level). In this paper, we call this set a cluster.

Here, we consider the process of the behavioral abstraction shown in Fig. 3, which shows that each upper level operation is constructed from lower level operations. For example, $M_2$ is synthesized from $C_3$, $C_4$ and $C_5$ that are sequentially executed in the same machine cycle. Assumed that these operations are given as Eq. (8), then $M_2$ is obtained

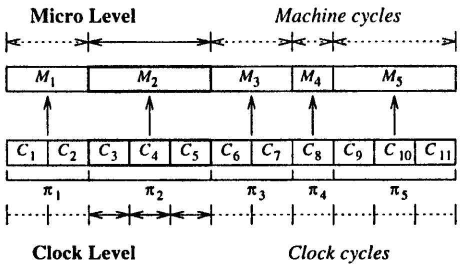


Figure 3: Example of behavioral abstraction

by the function composition for the cluster $\pi_2$:

$$\begin{aligned}[M_2] &= [C_3]\circ[C_4]\circ[C_5], \qquad (12)\\ \pi_2 &= \{[C_3],[C_4],[C_5]\},\end{aligned}$$

where $\circ$ denotes the function composition.

From the above reason, the behavioral abstraction is defined as the function composition for a cluster $\pi$:

$$\begin{aligned}[U] &= [L_1]\circ[L_2]\circ\cdots\circ[L_K] \\ &= [L_K](\cdots([L_2]([L_1]))\cdots), \qquad (13)\\ \pi &= \{[L_1],[L_2],\cdots,[L_K]\},\end{aligned}$$

where $[U]$ and $[L_k]$ $(k = 1, 2, \cdots, K)$ denote functions that correspond to primary operations at an upper level and a lower one, respectively.

The behavioral abstraction can therefore be carried out automatically by the trace table [3], a technique for the function composition.

### 4.2 Temporal abstraction

We consider again an example in Fig. 3. Figure 4 shows a corresponding example of the temporal abstraction. In Fig. 4, each circle denotes a time tick that indicates the starting point of each primary operation. Besides, each numerical label of circles indicates the order of these operations. The arcs indicate the relation between time ticks at the upper level and those at the lower one. This relation called the temporal abstraction function, $\mathcal{F}$, which generate the numerical labels at the upper level (e.g., micro level) from those at the lower one (e.g., clock level).

This function is given for the general case as the following form [11]:

$$\begin{aligned}&\forall\tau_a, \tau_b \in \mathcal{N}, \tau_a > \tau_b \ \Rightarrow \\ &\qquad \mathcal{F}(\mathcal{G}(\tau_a), \tau_a) \ > \ \mathcal{F}(\mathcal{G}(\tau_b), \tau_b), \qquad (14)\end{aligned}$$

where $\mathcal{N}$ is a set of natural numbers and $\mathcal{F} \in \mathcal{N}$. In Eq. (14), $\mathcal{G}$ is a predicate to indicate the starting

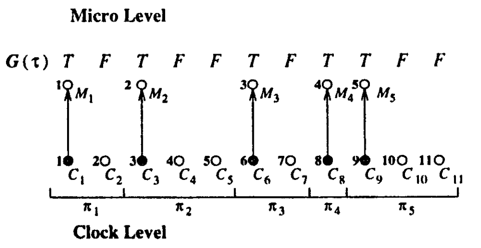


Figure 4: Example of temporal abstraction

points at an upper level. In addition, $\mathcal{F}$ is defined if and only if $\mathcal{G}$ is true.

Since Eq. (14) has enough degree of freedom to be selected, we cannot directly apply the function $\mathcal{F}$ to generating automatically the start points and the numerical labels. We therefore give the function $\mathcal{F}$ for the particular case to generate the starting points and the numerical labels.

Firstly, we consider an example in Fig. 4 to decide the predicate $\mathcal{G}$. Each starting point is generally selected as the time tick whose numerical label is the smallest in a cluster. Therefore, $\mathcal{G}$ is obtained for a cluster $\pi$ as:

$$\mathcal{G}(\tau) \Leftrightarrow \tau = \min_{\forall [L_k] \in \pi} (get_\tau([L_k])), \qquad (15)$$

where $\tau \in \mathcal{N}$, $get_\tau(\cdot)$ indicates the numerical label of its argument and $L_k$ is the same in Eq. (14).

Secondly, in order to give the numerical label at an upper level as incremental series (e.g. $1, 2, 3, \cdots$), we impose the following conditions on the function $\mathcal{F}$:

$$\begin{aligned} \mathcal{G}(1) &= true, & (16)\\ \mathcal{F}(\mathcal{G}(1), 1) &= 1 & (17) \end{aligned}$$

and

$$\mathcal{F}(\mathcal{G}(a), a) = \mathcal{F}(\mathcal{G}(c), c) + 1 \qquad (18)$$

if and only if $\forall a, b, c \in \mathcal{N}$

$$\begin{aligned} & \mathcal{F}(\mathcal{G}(a), a) < \mathcal{F}(\mathcal{G}(b), b) < \mathcal{F}(\mathcal{G}(c), c) \\ \wedge \ & \mathcal{F}(\mathcal{G}(b), b) \notin \mathcal{N}. \qquad (19) \end{aligned}$$

Finally, by applying Eq. (15) to Eq. (14) with the conditions, Eq. (16)–(19), the starting points of the upper level operations and their order can be generated as incremental series.

### 4.3 A generating algorithm

By using the above abstraction methods, we have the algorithm to generate HSMAP:

**Step 1)** The abstraction level of HSMAP is selected according to the simulation cycle required for the purpose of simulation. The specification *RTL* has this information.
**Step 2)** From *RTL*, the clusters are generated as the following form:

$$\Pi = \{\pi_1, \pi_2, \cdots, \pi_N\}.$$

**Step 3)** To generate the primary operations and their starting points, Eq. (13) and Eq. (15) are applied to the clusters $\pi_n$ $(n = 1, 2, \cdots, N)$.
**Step 4)** To generate the order of the above operations, Eq. (14) is applied to the results in Step 3).

From the above algorithm, we can get HSMAP defined in Eq. (1).

## 5 Experimental Results

We have implemented a prototype system in Perl on a Sun SPARC Station 20. Then we have made an experiment and applied proposed methods to thirty instructions of *DLX* processor [4]. In this experiment, we consider generating each simulation model at the instruction level, the micro level and the clock level from the phase level specification.

Figure 5 shows a part of this specification, where the register transfers are divided into groups. These groups are made according to each regular interval at the level of abstraction: In this experiment, we have selected an instruction cycle, a machine cycle and a clock cycle.

In Table 2, we show the CPU time required for generating each of the above three simulation models. From Table 2, we see that the CPU time is less than 30 sec. for thirty instructions of DLX processor.

```
ADD:  IF:   1:  MAR := PC;
            2:  IR := MEM(MAR);
      ID:   3:  PC := +(PC,4),
                A := RF(IR<15:11>),
                B := RF(IR<20:16>);
      EX:   4:  TMP := B;
            5:  C := +(A,TMP);
      MEM:
      WB:   6:  RF(IR<10:6>) := C;
```

Figure 5: Example of the specification

## 6 Conclusions

In this paper, we proposed two abstraction methods for generating each simulation model at different levels of abstraction; then provided its generation algorithm. Furthermore, we showed these

Table 2: Experimental results on Sparc Station 20

| Level of Abstraction | CPU time (sec.) |
|---|---|
| Instruction Level | 29.6 |
| Micro Level | 24.8 |
| Clock Level | 15.2 |

methods were implemented as the function composition based on the functional representation of register transfers. Finally, applying these methods to DLX processor, we confirmed that the simulation model was generated in less than 30 sec. for 30 instructions.

## References

[1] Anceau,F.: *The Architecture of Microprocessors.*, Addison-Wesley Publishing Company, (1986).

[2] Zelkowitz,M.V.: "A Functional Correctness Model of Program Verification", *IEEE Computer*, **vol.23**,No.11, (Nov 1990).

[3] Gannon,J.D., Hamlet,R.G., Mills,H.D.: "Theory of Modules", *IEEE Trans. on Software Engineering*, **vol.SE-13**,No.7, (July 1987).

[4] Hennessy,J.L, Patterson,D.A: *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers Inc. (1990).

[5] Wolf,W.H.: "Hardware-Software Co-Design of Embedded Systems", *Proceedings of the IEEE*, **vol.82**,No.7, pp.967-989, (July 1994).

[6] Gong,J., Gajski,D.D., Nicolau,A.: "A Performance Evaluator for Parameterized ASIC Architectures", *Proceedings of the European Conference on Design Automation*, (1994).

[7] Schuette,M.A.,Shen,J.P.: "Exploiting Instruction-Level Parallelism for Integrated Control- Flow Monitoring", *IEEE Trans. on Computers*, **vol.43**,No.2, pp. 129-140, (February 1994).

[8] Kalavade,A., Lee,E.A.: "A Hardware-Software Codesign Methodology for DSP Applications", *IEEE Design & Test of Computers*, **vol.10**,3, pp.16-28, (September 1993).

[9] Diep,T.A., Shen,J.P., Phillip,M.: "EXPLORE: A Retargetable and Visualization Based Trace-Driven Simulator for Superscalar Processors", *Proceedings of the 26th Annual International Symposium on Microarchitecture*, pp. 225-235, (1993).

[10] Baudendistel,K.: "CLIFF: C Language Interface for the Functional Simulator", *Proceedings of the ICASSP*, pp. 3263-3266, (1995).

[11] Windley, P.J.: "Formal Modeling and Verification of Microprocessors", *IEEE Trans. on Computers*, vol.44,No.1, pp. 54-72, (January 1995).

[12] Tahar,S.,Kumar,R.: "Towards a Methodology for the Formal Hierarchical Verification of RISC Processors", *Proceedings of the ICCD*, pp. 58-62, (1993).

[13] McCrackin,D.C.,Srinivasan,S.: "Trace Driven Pipeline and Cache Simulation of Multithreaded Computers", *Simulation*, (August 1994).