Instructional Scaffolding for ASIP Design Education with System Verilog Assertion considering Situated Nature of Learning

Ryuichi TAKAHASHI† **and Yoshiyasu TAKEFUJI**††,

Hiroshima City University, Keio University

Summary
This paper proposes a new method for bringing out challenges to study modern microprocessor design at the graduate course from an advanced senior project for application specific instruction set processor (ASIP) design using SystemVerilog assertion with related design verification technologies. We have already reported that the instruction issue logic for superscalar microprocessors works very well as an way-in of legitimate peripheral participation (LPP) to observe the system, since the instruction issue logic is tuned at the final stage of the design and is the central part of the system. The way-in gave opportunity for students to understand the mechanism of superscalar microprocessors. The fine grain microprocessor design education for junior students has been greatly improved. Our next step is to find more general solution to the subject to improve our microprocessor design education, since the instruction issue logic seemed to be too specialized for particular microprocessor design. The answer was SystemVerilog assertion (SVA) describing central part of the system with related model checking methodologies which are theoretical successor to the SVA. A senior student who designed a pipelined ASIP for Dijkstra's algorithm as a senior student continued to study an extension of abstraction technique to investigate the behavior of the algorithm on the flowchart beside the Kripke structure for identifying the location of the bugs as a graduate study for his master's thesis. Another senior student who designed another pipelined ASIP for Kruskal's algorithm as a senior student continued to study derivation of sophisticated temporal logic equations from simple properties proved by simple model checking using inference rules with theories as his graduate study. We believe showing the central part of the system by using SVA with related technologies for the design verification will work very well for guiding the undergraduate students to learn the mechanism of the system as well as advanced studies of modern microprocessor design at the graduate course.

Key words:
ASIP, SVA, model checking, design verification, LPP

1. Introduction

The microprocessor design education has become one of the very important subjects in this time of upheaval. An application specific instruction set processor (ASIP) is one of the key solutions used for system-on-a-chip design to provide a tradeoff between the flexibility of a general purpose CPU and performance of an application specific integrated circuit (ASIC) for a special purpose to reduce the cost and power consumption Digital signal processors with multiply–accumulate (MAC) operations can be regarded as a typical ASIP. For designing ASIPs, the mechanism of computation must be understood well for the first step.

Instructional scaffolding[1] is a teaching method introduced by Lev Vygotsky who believed that when student is in the zone of proximal development (ZPD) for a particular task, providing an appropriate assistance will give the student enough of boost to achieve the task. Constructive approach[2] could be one of a good solution to the subject for guiding the students to understand how computer works. Twelve subjects are given to show the elements of computer built from scratch. Our approach takes into account the sequence to show the design phases. First, we give an opportunity to observe the outline of system by using "way-in" of LPP[3]. We extended the LPP by choosing the central part of the system as the way-in[4]. We used the instruction issue logic for superscalar microprocessor design as the way-in for the extended LPP, since the logic is tuned at the final stage of the design and is the central part of the system. Many devices have become appeared among the designs by junior students. We have been tried to find more general solution to the subject for improving our microprocessor design education, since the instruction issue logic seemed to be too specialized for superscalar microprocessor design.

In 1990's, design on hardware description language (HDL) has become daily routine for digital system design. Design verification is expected to be the next innovation in the field of electronic design automation (EDA). SystemVerilog[5] is a superset of an hardware description language Verilog HDL, which we have been using for our microprocessor design education. Properties that should be held in the system could be written as SystemVerilog assertion (SVA), which can be used to describe the heart of the system as the way-in of LPP. Furthermore, design verification is done at the very final stage of the microprocessor design. With the SVA, showing model checking[6] methodologies seemed to be worked as a good guidance for the students to the further research at the graduate course, since the model checking is the theoretical successor to the SVA.

The next section describes the prior microprocessor design education which seemed to be too specialized for the superscalar microprocessor design. Section 3 describes the

Manuscript received October 5, 2016
Manuscript revised October 20, 2016

design verification by SVA and model checking methodologies shown to the students after the ASIP design. Section 4 describes the studies accomplished by the students after the ASIP design and the guidance, where linear temporal logic (LTL) and computation tree logic (CTL) are introduced as a typical way to describe the behavior on Kripke structure.

2. Prior microprocessor design education

The legitimate peripheral participation (LPP) [3] has been introduced by Lean Lave and Etienne Wenger, who noticed the importance of the situated nature of learning. They investigated the tailors in West Africa. The steps of the apprenticeship were reversed production steps, which have effect of focusing the apprentices' attention first on the broad outline of the product construction. The apprentices begin by learning the finishing stages of producing a garment, go on to learn to sew it, and only later learn to cut it out. Each step offers the opportunity to consider how the previous step contributes to the present one. In addition, this ordering minimizes experiences of failure and especially of serious failure.

We paid attention to the fact that the learning steps are reversed production steps. This is for the learners to have an opportunity to get the broad outline of the product without having serious risk "Way-in" refers to period of observation and attempts to construct a first approximation of the garment. We noticed that we should use a subject treated at the final stage of the design phases. We also noticed that if we could find a subject which is the heart of the product, the subject will be a very good way-in for the LPP. This is our extension. The subject for the way-in is desirable to be central part of the product.

The answer was the instruction issue logic for the superscalar microprocessors[4]. The logic is tuned at the final stage of the design phases and is the central part of the superscalar microprocessor. Before teaching superscalar microprocessor design, pipelining had been instructed. Pipelining was too difficult, since the division of the modules for the pipelining was "cutting" at the first step of the design. The pipelining education was a failure. The average number of the lines of the modification was 127 and the standard deviation was only 162 lines. Few students succeeded to introduce their original devices except internal/external interruption handling. Moreover the completion ratio was 80%.

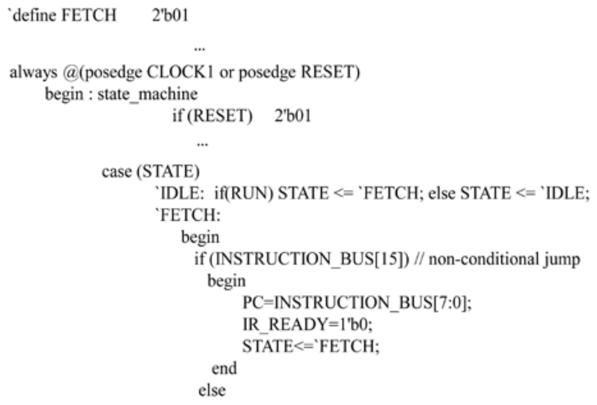
After we started superscalar microprocessor design education showing the role of the instruction issue logic, many devices have become appeared among the designs by junior students. In 2007, which is the 3rd year of our superscalar microprocessor design education, 50 out of 53 junior students succeeded to complete the entire design and fabrication phases. The average number of the lines of the modification was 407 and the standard deviation was 975 lines. 20 out of 53 students succeeded to introduce their original own devices in addition to the simple internal/external interruption handling. The remarkable point is the fact that the students have begun to use their head to create their own original designs. They could have an opportunity to understand the very mechanism of the superscalar microprocessor, since the way-in was the central part of the product.

Our next step is to find more general solution, since the instruction issue logic for superscalar microprocessor seemed to be too specialized for particular microprocessor design.

We noticed that SystemVerilog assertion (SVA) could be used to describe the central part of the system at the very final stage of the design phases. SystemVerilog[5] is a hardware description language backward compatible with Verilog HDL.

In our microprocessor design education, Verilog HDL has been used to describe the microarchitecture of the microprocessors. Register transfer level behavior is written as state machines for the processors. The state machine for the prefetch unit only continues to fetch the instructions one after another. Between the prefetch unit and decode unit, there exists an instruction register as a staging latch.

Figure 1 illustrates a part of the description of the prefetch unit in Verilog HDL. For each state machine, register transfer operations are written with state transitions in individual state of the machine in Verilog HDL like a classical hardware description language DDL[7] proposed by Duley and Dietmeyer in 1968.



```
`define FETCH      2'b01
                       ...
always @(posedge CLOCK1 or posedge RESET)
      begin : state_machine
                        if (RESET)    2'b01
                        ...
          case (STATE)
               `IDLE:  if(RUN) STATE <= `FETCH; else STATE <= `IDLE;
               `FETCH:
                   begin
                     if (INSTRUCTION_BUS[15]) // non-conditional jump
                       begin
                            PC=INSTRUCTION_BUS[7:0];
                            IR_READY=1'b0;
                            STATE<=`FETCH;
                       end
                     else
```

Fig. 1 A state machine description in Verilog HDL

Programs for the processors are written in binary image as the contents of the instruction memory. Data to be processed are described in data memory.

3. Verification technologies

We paid attention on SystemVerilog's ability to describe properties which is expected to be held in the system. The properties could be written in SystemVerilog assertion (SVA) separately with the original description of the system in Verilog HDL.
The SVA could write "condition" that should be held always like:
assert property ($onehot0(select[15:0]);
describing that only one output signal of a selector could be one at the same time. This could be regarded as a description in propositional logic.
An SVA description "response" like
assert property (A==B |=> C ##1 D);
describes "if A equals to B, C must be one in the next clock cycle and D should be one after one more clock cycle later." "|=>" is called non-overlap implication, which separates the clock cycles.
assert property ($rose(REQ)|=> ##[0:$] ACK);
describes "if signal REQ rose, signal ACK will be one finally." These could be regarded as description in temporal logic for Kripke structure S(W,R,V) , where W is a set of the states, R is a reachability relation among the states and V is the assignment of the logical values to the logical variables in each state.
We instructed assertion based verification (ABV) in our senior project to design ASIPs on Verilog HDL with SystemVerilog. The students observed their designs to write down the key properties in SVAs, which are the new way-in in our new trial. SVAs worked very well as a new way-in, since the students have to write down the heart of the system in SVA to verify the behavior of the ASIPs. SVAs also worked very well for finding bugs in short time. Students completed their ASIP designs for their particular applications in a couple of months in their senior student days.
After designing ASIPs as senior students, the students are instructed model checking[6] methodologies in the first half year of their graduate school days.
Fig. 2 illustrates the relation between ABV and model checking. Properties written in SVA correspond to the properties described in temporal logic.

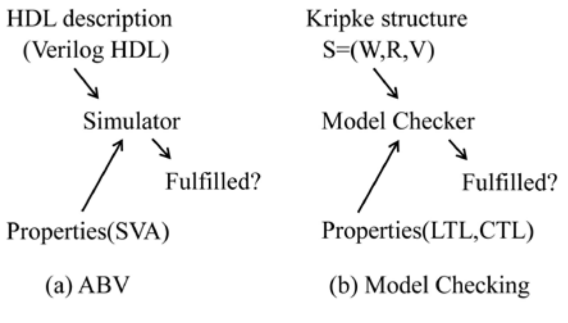


Fig. 2 Relation between ABV and Model Checking

In ABV, only some cases are investigated, since the verification is accomplished through simulation. In model checking, all possible cases are investigated on Kripke structure.
In LTL, properties are described for a particular computation path of the state transition on Kripke structure. A state of a system satisfies an LTL formula if all paths from the given state satisfy it. Abstraction is recognized as one of the solutions to the state explosion problem. Abstraction mapping from concrete structure C to abstract structure A is discussed. The state of traffic signal C={Green, Yellow, Red} could be mapped to A={Green, Not-green}. Data mapping is for a particular variable. Predicate abstraction is for variables more than one.
In CTL, as well as the temporal operator U(Until), F(some Future state), G(Globally) and X(neXt state) of LTL, quantifiers A(All paths) and E(Exists a path) are used. Existence of some particular pipeline hazard condition could be treated in CTL.
Model checking[5] methodologies were instructed by using NuSMV[8],[9]. NuSMV is a symbolic model checker, where both LTL and CTL could be used. Several examples are asked to be investigated for the graduate students by using NuSMV to understand the model checking methodologies after designing ASIPs using ABV.
An application of theorem proving to formal design verification is also introduced besides the model checking. In the application, designs are expressed as axioms and the properties are expressed as the theorems to be proved. We also instructed that there is a limitation since the theorem proving depends on particular notation for the theorem proving system.

4. Studies accomplished by students

Two senior students designed their own ASIPs by modifying an example description of a pipelined RISC having 3 stages. Theexample description is written in 498 lines of Verilog HD and verified by ABV with SVA.
Fig.3 illustrates an assertion example in SVA to check a forwarding mechanism for the pipelined RISC. When the state of prefech unit changed to "FETCH" from "WAIT" for pipeline interlock, forwarded value of the flag register must be equal to the latest value of the flag register "FR"

```
flag_register_forward: assert property
   ( @(posedge CLOCK1)
     (STATE==`WAIT) ##1 (STATE==`FETCH)|->
          $past(FR_FW) == FR );
```

Fig.3 An assertion example for pipelining

Pipeline interlock is a key mechanism for pipelining. The heart of the system is now shown to the students through

SVA, which is more general in comparison with instruction issue logic for superscalar microprocessors.
A senior student designed an ASIP for Dijkstra's algorithm to find a shortest path by modifying the example RISC description. The ASIP was written in 1063 lines of Verilog HDL, which could be implemented by using 454 LUTs of Xilinx FPGA.
Fig. 4 illustrates the block diagram of the ASIP. Give graph is expressed as linked list on the data memory. In the execution unit, 3 out of 4 index registers were used to handle the linked lists expressing the weighted graphs for the Dijkstra's algorithm.

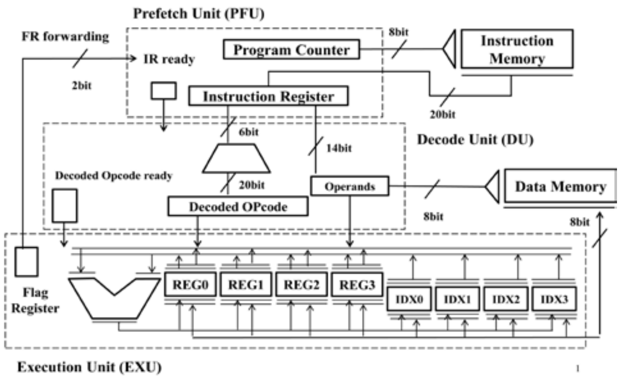


Fig. 4 An ASIP for Dijkstra's algorithm

The Dijkstra's algorithm is expressed in binary image as the contents of the instruction memory, while the given graph is expressed in binary image as the contents of the data memory in Verilog HDL
Fig. 5 illustrates the flowchart for the Dijkstra's algorithm. Labels L1,L2,…are implemented by program counter (PC). The senior student tried to find the location of the bugs by checking each segment of the flowchart.

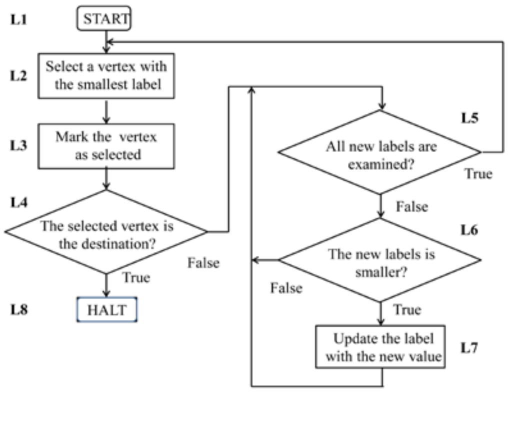

L1
START
L2
Select a vertex with the smallest label
L3
Mark the vertex as selected
L4
The selected vertex is the destination?
True
False
L8
HALT
L5
All new labels are examined?
True
False
L6
The new labels is smaller?
False
True
Update the label with the new value
L7

Fig. 5 A flowchart for Dijkstra's algorithm

Fig. 6 illustrates an assertion example to check a segment.

When he found an SVA failed, he divided the segment into shorter segments for identifying the location of the bug. A bug on software in binary image as the contents of the instruction memory was found. This could happen because the algorithm was implemented not only by the hardware but also by the software. System verification has been done by the SVA.

```
label_compare: assert property
  ( @(posedge CLOCK1)
    PC==L5 |-> $stable(MEM[IDX1+4])
          || $past(MEM[IDX1+4]) > MEM[IDX1+4]
          || $past(PC)==L4 );
```

Fig.6 An assertion example for Dijkstra's algorithm

After designing the ASIP and the exercise using NuSMV, the student noticed that the predicate abstraction for LTL could be used to investigate the segments of the flowchart.

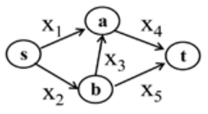


Fig. 7 An example graph

Fig. 7 illustrates a graph for the Dijkstra's algorithm, where 3 cases exit as the shortest path from starting vertex s to terminal vertex t.
There are 3 types of paths on the ASIP. First one is the very shortest path from starting vertex to the terminal. There are 3 cases in this example: s->a->t, s->b->a->t and s->b->t. The second one is the path on Kripke structure, where the initial state is the moment when all vertexes are labeled infinite and none of the vertexes are selected. The states are formed by the contents of the PC, the value of the labels and marks indicating if the vertexes are selected. The final state on the path is the moment when the terminal vertex is selected. Although each case of the shortest path on the graph in Fig. 6 could not identify particular path on the Kripke structure, a predicate
$x_2<x_2+x_3<x_1<x_2+x_3+x_4<x_1+x_4<x_2+x_5$ (1)
identifies a path on the Kripke structure. Each vertex could be visited more than one time by the algorithm for a particular shortest path. The third type is the path of the flowchart characterized by the contents of the PC. For the path specified by the inequality (1), the program segment L5->L6->L7->L5 is traversed 5 times on the flowchart, which can be verified by ABV. He found a way to use predicate abstraction to reduce the number of the states also could be used for investigating the behavior of the algorithm on the flowchart. He selected appropriate numbers(weights) which satisfy the inequality (1) to check

the segement L5->L6->L7->L5 is traversed exactly 5 times by ABV.
Another senior student designed an ASIP for Kruskal's algorithm to find a maximum spanning tree by modifying the example RISC description. The ASIP was written in 1468 lines of Verilog HDL, which could be implemented by using 991 LUTs of Xilinx FPGA.
Fig. 8 illustrates the block diagram of the ASIP. The unit indicated by A in the figure is a special state machine for adding an edge to the current set of edges which does not contain any loop (cycle).

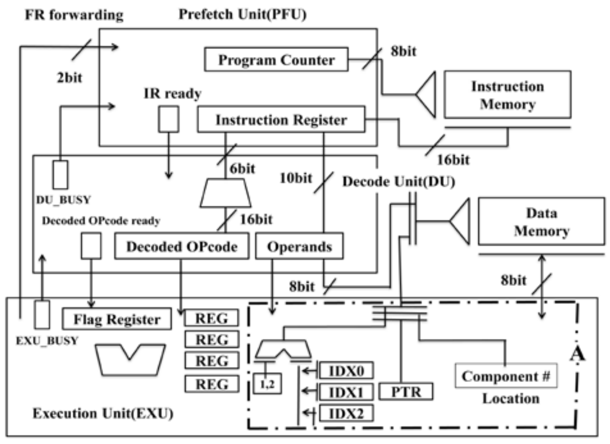


Fig. 8 An ASIP for Kruskal's algorithm

Kruskal's algorithm is a greedy strategy, which can produce an optimum result when and only when M(E,I) is a matroid, where E is a set of edges and I is the subset of 2^E without loop in this case.
Fig. 9 illustrates an assertion example to check if a connected component has no loop, where V is the number of the vertexes and E is the number of the edges. The following two statements are equivalent:

(1) A graph G is a tree
(2) A graph G is connected and V=E+1.

The SVA was checked for each connected component always when a new edge is added to the current set of edges.

```
core_tree1: assert property
  ( @(posedge CLOCK1)
    ADD_EDGE && V && E |=> (V==E+1) );
```

Fig.9 An assertion example for Kruskal's algorithm,

After designing the ASIP and the exercise using NuSMV, the student noticed that the above equivalence (theorem) could be used as a premise for reasoning to prove properties. On the other hand, simple properties required could be proved by some simple model checking.

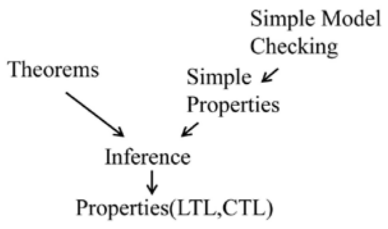


Fig. 10 Property checking by influence

Fig. 10 illustrates the idea to derive properties by inference from theorems and simple properties, which could be proved easily by appropriate model checking..
Fig 11 illustrates an example of such an inference rule[10], where "⊦" indicates the existence of the proof.

$$\frac{\vdash f \Rightarrow g \quad \vdash f \Rightarrow \mathbf{X}\, f}{\vdash f \Rightarrow \mathbf{G}\, g}$$

Fig. 11 An example inference rule

In Fig.11, "f" could be a statement saying graph G is connected and V=E-1 while "g" could be a statement saying G is connected without having any loop. This is a theorem. "f ⇒Xf," where "Xf" means that f is true at the next state, could be proved by some model checking. We can conclude that if "f" is true at some state "g" is true globally.
Constructing an appropriate set of inference rules which has soundness and completeness seemed to be a future problem.

5. Conclusion

A new method for bringing out challenges to study modern microprocessor design at the graduate course from experiences to design ASIPs using SVA with related design verification technologies in senior student days is proposed. The key point is to show the heart of the system with SVA with methodologies for advanced verification. The method should be suitable along with the progress of verification technologies in the field of EDA.

Acknowledgments

The authors would like to thank graduate students Tomohisa Fujita and Masakazu Hirakawa for their participation in this experiment. This work is supported by VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with Cadence Design Systems, Inc.

References

[1] Roland G. Tharp and Ronald Gallimore: Rousing minds to life: Teaching, learning, and schooling in social context, Cambridge university press, 1988

[2] Noam Nisan and Shimon Schocken: The Elements of Computing Systems, The MIT Press, 2005

[3] Jean Lave and Etienne Wenger, Situated learning, Legitimate peripheral participation. Cambridge university press, 1991.

[4] Ryuich TAKAHASHI, Hajime OHIWA and Yoshiyasu TAKEFUJI "A Fine Grain Microprocessor Design Education considering Situated Nature of Learning," International Journal of Computer Science and Network Security, Vol.8 No.6, pp.194-198,2008

[5] Stuart Sutherland, Simon Davidmann, Peter Flake forword by Phil Moorby: SystemVerilog for Design, Kluwer Academic Publishers, 2004

[6] Edmund M. Clarke, Jr., Orna Grumberg, and Doron A. Peled: Model Checking, The MIT Press, 1999

[7] James R. Duley and Donald L. Dietmeyer: "A Digital System Design Language (DDL)," IEEE Trans. Computers, vol. C-17, No.9, pp.850-861, 1968

[8] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani and A. Tacchella:"NuSMV 2: An OpenSource Tool for Symbolic Model Checking," Proceedings of International Conference on Computer Aided Verification. pp.359-364, 2002

[9] Michael Huth and Mark Ryan: Logic in Computer Science second edition, Cambridge university press, 2004

[10] Ming-Hsien Tsai and Bow-Yaw Wang: "Formalization of CTL* in Calculus of Inductive Constructions," ASIAN 2006, LNCS4435, pp.316-330, 2007

Ryuichi Takahashi received the B.S. degree in physics from Waseda University in 1978 and M.E. degree in information processing from Tokyo Institute of Technology in 1981. He received the PhD degree from Keio University in 2010 with a thesis entitled "Microarchitecture Education using the Design on HDL," where the letters HDL stand for hardware description language.. During 1981-1991, he worked for NEC Corp. as a researcher as well as a VLSI engineer. In 1991, he moved to Tokyo Institute of Technology, where he had been having a class for microcomputer design using TTL as an assistant professor.. He joined Hiroshima City University in 1994, where he is currently an associate professor on faculty of information sciences. He received excellent educator award from Information Processing Society of Japan (IPSJ) in 2004 for his educational activity known as City-1 He published a couple of books on Verilog HDL and digital algebra.

Yoshiyasu Takefuji is a tenured professor on faculty of environmental information at Keio University since April 1992 and was on tenured faculty of Electrical Engineering at Case Western Reserve University since 1988. Before joining Case, he taught at the University of South Florida for 2 years and the University of South Carolina for 3 years. He received his BS (1978), MS (1980), and Ph.D. (1983) in Electrical Engineering from Keio University. His research interests focus on neural computing, security, electronic toys. He received the National Science Foundation Research Initiation Award in 1989, the distinct service award from IEEE Trans. on Neural Networks in 1992, the TEPCO research award in 1993, the Takayanagi research award in 1995, the Kanagawa Academy of Science and Technology research award in 1993, the best courseware award from Asia multimedia forum in 1999, the best paper award of Information Processing Society of Japan in 1980, special research award from the US air force office of scientific research in 2003, chairman award from JICA in 2004. He authors 25 books including neural network parallel computing in 1992, and has published more than 200 papers.