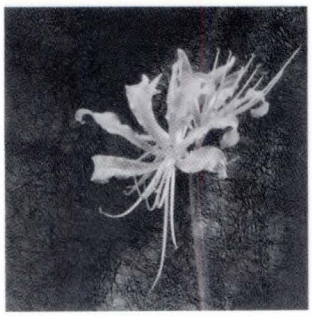
ISSN 0016-2523

FUJITSU

SCIENTIFIC & TECHNICAL JOURNAL

Summer 1993 VOL. 29, NO. 2



The Issue's Cover :

盛花 (Flowering)

FUJITSU Scientific & Technical Journal is published quarterly by FUJITSU LIMITED of Japan to report the results of research conducted by FUJITSU LIMITED, FUJITSU LABORATORIES LTD., and their associated companies in communications, electronics, and related fields. It is the publisher's intent that FSTJ will promote the international exchange of such information, and we encourage the distribution of FSTJ on an exchange basis. All correspondence concerning the exchange of periodicals should be addressed to the editor.

FSTJ can be purchased through KINOKUNIYA COMPANY LTD., 38-1, Sakuragaoka 5-Chome, Setagaya-ku, Tokyo 156, Japan, (Telephone : +81-3-3439-0162, Facsimile : +81-3-3706-7479).

The price is US$7.00 per copy, excluding postage.

FUJITSU LIMITED 1015 Kamikodanaka, Nakahara-ku,
Kawasaki 211, Japan
Cable Address : FUJITSULIMITED KAWASAKI
Telephone : +81-44-777-1111
Facsimile : +81-44-754-3562

Printed by MIZUNO PRITECH Co., Ltd. in Japan

FUJITSU

SCIENTIFIC & TECHNICAL JOURNAL

Summer 1993 VOL. 29, NO. 2

CONTENTS

Papers

UDC 519.68:681.326

ParaGraph: A Graphical Tuning Tool for Multiprocessor Systems

● Seiichi Aikawa ● Mayumi Kamiko ● Takashi Chikayama

(Manuscript received November 30, 1992)

Distributing computational load to many processors is a critical issue for efficient program execution on multiprocessor systems. Finding a good load distribution algorithm is one of the most important research topics for parallel processing. Tools for evaluating load distribution algorithms are very useful for this kind of research. This paper describes a system called ParaGraph that gathers periodical statistics of the computational and communication load of each processor during program execution, in both the higher level of programming language and lower level of implementation, and presents them graphically to the user.

1. Introduction

In the Japanese Fifth Generation Computer Systems Project, parallel inference systems have been developed for promoting parallel software research and development. The system adopts a concurrent logic programming language KL1[1] as the kernel and consists of a parallel inference machine, PIM[2] and its operating system, PIMOS[3].

For efficient program execution, the computational load must be appropriately distributed to each processor. On scalable loosely-coupled multiprocessor systems, load balancing and minimization of communication overhead are essential, but become more difficult compared to tightly-coupled systems as communication costs increase. Although many load distribution algorithms have been developed[4, 5], none have been sufficient to execute every program effectively. Finding a good load distribution algorithm is one of the most important research topics for parallel processing.

Tools for evaluating load distribution algorithms are very useful for this kind of research. The objective of the ParaGraph system is to help programmers design and evaluate load distribution algorithms on loosely-coupled multiprocessor systems. ParaGraph gathers profiling information during program execution on the parallel inference machine, PIM, and displays it graphically based on the X window system[6].

Many performance displays have been devised for utilization, communication, and task information[7, 8]. For example, graphical meters[7] represent processor-utilization and graphical animation on a processor configuration map[8] represents interprocessor-communication of message-passing programs. Such specialized views provide an intuitive feeling for dynamic behavior, but it is difficult to determine where the performance bottlenecks are. Because the execution of parallel programs often raise complex phenomena, simple observation of each phenomena can not provide full information needed to detect performance bottlenecks. For example, suppose that when tasks are not mutually independent and must communicate with each other closely. The program is less efficient because of communication overhead. But graphical meters may show processors work hard, although most of processing time must have been consumed on message-handling. In this case, it is useful to compare the activity of processors with frequencies of sending and receiving messages along execution time. Thus, bottlenecks are often determined by comparing

with some pieces of profiling information each other. In ParaGraph system, every kind of profiling information can be displayed based on three common axes to be easy to compare. Because such profiling information can be viewed as having three axes: what, when, and where.

In chapter 2, how load distribution can be described in KL1 on PIM are described. Chapter 3 describes the implementation of the ParaGraph system and graphical representation of program execution, and chapter 4 discusses how useful graphical displays are to detect performance bottlenecks with examples of various programs.

The contents of this paper partially overlap the subject of a previous paper[9].

2. Load distribution algorithms

2.1 Load distribution in KL1

The parallel inference machine runs a concurrent logic programming language called KL1[1), 3), 10)]. A KL1 program consists of a collection of guarded Horn clauses of the form:

$H:-G_1,...,G_m \mid B_1,..., B_n$ ($m, n \geqq 1$),

where H, G_i, and B_i are atomic formulas. H is called the head, G_i, the guard goals, and B_i the body goals. The guard part consists of the head and the guard goals and the body consists of body goals. They are separated by the commitment operator " | ". A collection of guarded Horn clauses whose heads have the same predicate symbol P and the same arity N, define a procedure P with arity N. This is denoted as P/N.

The guard goals wait for instantiations to variables (synchronization) and test them. When the guard part of one or more clauses succeed, one of those clauses is selected and its body goals are called. These body goals communicate with each other through their common variables. If variables are not ready for testing in the guard part because the value has not been computed yet, testing is suspended.

In addition to the above basic mechanism, there is a mapping facility which includes load distribution specification. The programmer can annotate the program by attaching pragmas to the body goals to specify a processor {specified by Goal@node (Proc)}. The programmer must tell the KL1 implementation which goals to execute on which processors.

```
next_queen(N,I,J,B,R,D,BL):- J>0, D=0 |
    BL = {BL0,BL1},
    R = {R0,R1},                    processor specification
    BL0 = [get(Proc)|BL2],                  ↓
    try_ext(N,I,J,B,R0,D,BL2)@node(Proc),
    next_queen(N,I,J-1,B,R1,D,BL1).
```

Fig. 1 – An example of a KL1 program.

Figure 1 shows a part of a KL1 program. If the goal next_queen/7 is committed to this clause, its body goals are called. The goal try_ext/7 has a processor specification, and it is to be executed on processor number "Proc". This processor number can be dynamically computed.

2.2 Design issues

Load balancing derives maximum performance by efficiently utilizing the processing power of the entire system. This is done by partitioning a program into mutually independent or almost independent tasks, and distributing tasks to processors. Many load balancing studies have been devised, but they are tightly coupled to particular applications. Therefore, programmers have to build load distribution algorithms for their own applications.

To distribute the computational load efficiently, the programmer should keep in mind the following points. Since load distribution is implemented by using goals, the programmer should understand the execution behavior of each goal. When goals are executed on a loosely-coupled multiprocessor, the programmer should investigate the load on individual processors and the communication overhead between processors.

For evaluating load distribution algorithms, tools must provide many graphic displays for the programmer to understand the computational and communication load of each processor in both the higher program and lower implementation levels. No single display and no single profiling level can provide the full information

needed to detect performance bottlenecks.

3. System overview

3.1 Gathering information

To statistically profile large-scale program execution, KL1 implementation provides information gathering facilities, low-level profiling and higher-level profiling. KL1 implementation provides these facilities as language primitives, to minimize the undesirable influence to the execution behavior of programs. These facilities have been implemented at the firmware level. The profiling facilities are summarized as follows.

1) Low-level profiling

Profiles the low-level behavior of the processor, such as how much CPU time went to the various basic operations required for program execution.

2) Higher-level profiling

Profiles the higher-level behavior of the processor, such as how many times each piece of the program was executed.

To minimize the perturbation, the gathered profiling information resides in each processor's local memory during program execution, and after execution, ParaGraph collects this information and converts into some standard form. Since profiling information is automatically produced by the KL1 implementation, programmers do not have to modify the application programs.

3.1.1 Low-level profiling

The basic low-level activities can be categorized into computation, communication, garbage collection, and idling. Computation means normal program execution such as goals' reductions and suspensions, communication means sending and receiving inter-processor messages, garbage collection means itself, and finally, idling means doing nothing.

The processor profiling facility measures how much time went to each category for each processor. Such information can be periodically gathered to show gradual changes of behavior. The profiling facility can also measure frequencies of sending and receiving various kinds of interprocessor messages[11), 12)].

1) A *throw_goal* message transfers a KL1 goal with a throw goal pragma to a specified processor.
2) A *read* message requests for some value from the remote processor when a clause selection condition requires it.
3) An *answer_value* message replies to a read message when the request value becomes available.
4) A *unify* message requests body unification (giving a value to a variable).

3.1.2 Higher-level profiling

KL1 provides a mechanism for grouping goals and controlling their execution in a meta-level. This mechanism can be considered to be an interpreter for the KL1 language. It also provides profiling facility at a higher level than processor profiling. Low-level profiling gathers a number of important statistics from many aspects that help analyzing performance bottlenecks, but it provides no information on where in the program is the root of such a behavior.

To correlate execution behavior with a portion of the program, higher-level profiling measures how many times goals associated with each predicate are reduced or suspended (due to unavailability of data required for reduction). Transition of behavior can be observed by periodically gathering the information.

3.2 Graphic displays

The profiling information can be viewed as having three axes: what, when, and where. In sequential execution, "where" is a constant and the "when" aspect is not important, since the execution order is strictly designated. Therefore, simple tools like gprof provided with UNIX[Note)] suffice. However, all three axes are important when parallel execution is concerned.

If such massive information is not presented carefully, the user might be more confused than informed. Therefore, ParaGraph provides graphic displays based on three axes. We named each representation using the terms "What,"

Note: The UNIX operating system was developed and is licensed by UNIX System Labiratories, Inc.

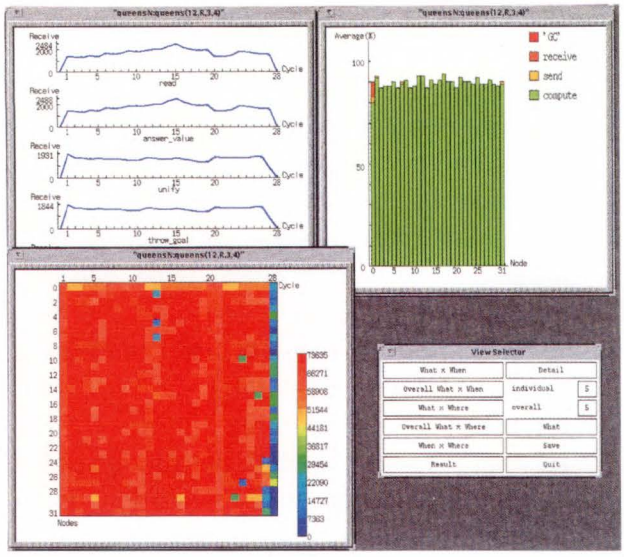


Fig. 2 – Examples of graphic displays: a What × When view (top-left), an overall What × Where view (top-right), and a When × Where view (bottom-left) and a menu-oriented user interface (bottom-left).

"When," and "Where." The term "What" is the visualization target corresponding to the type of profiling information such as low-level processor behavior, higher-level processor behavior, and interprocessor message frequencies. The term "When" and "Where" indicate time expressed by a cycle number and the processor number respectively.

Figure 2 shows the graphic displays of ParaGraph. These displays are execution behavior of all solution search program of N queen problem.

Every type of profiling information can be easily displayed with the views described below with a menu-oriented user interface such as the bottom-right window in Fig. 2. If the window size is too small to display everything in detail, coarser display aggregating several cycles or several processors together is possible to see the overall behavior at a glance. Scrolling on the vertical and horizontal directions are also possible if details are to be examined. It is also possible to display only selected "What" items.

3.2.1 A What × When view

There are two kinds of views in terms of "What" and "When" items. One is a What × When view which shows the behavior of each "What" item during execution. A graph is displayed of a "What" item in order of the total volume. The x axis is the cycle numbers, and the y axis is the rate of processor utilization, the number of messages, and the number of reductions or suspensions corresponding to the type of profiling information. Since every graph is drawn with the same scale on the vertical axis, it is easy to compare with "What" items.

The other is an overall What × When view which shows the behavior of all "What" items during execution. Each "What" item is stacked in the same graph and displayed by a line. The y axis represents the average rate of processor utilization, the total number of messages, and the total number of reductions and suspensions corresponding to the type of profiling information.

These views are helpful for example, if a program has sequential bottlenecks such as tight synchronization. In this case, the number of goal reductions will be down at some portion during program execution. Such a problem will be detected easily by observing program execution.

The top-left window in Fig. 2 shows received message frequencies on all processors with What × When view. In this window, four kinds of receiving message frequencies are displayed on each graph. These messages are displayed in order of the total number of received messages. The other messages are displayed by scrolling vertically.

From this, we know that each received message frequency on all processors is less than 2 500 times/an interval (an interval is 2 second). As this program is divided mutually independent subtasks, communication message frequency is very low.

3.2.2 A When × Where view

A When × Where view shows the behaviors of all "What" items on each processor. Each processor is displayed with various color patterns that indicate volume. The relationship between color patterns and volume are shown in the bottom right corner. The brighter the pattern, the busier the processor. Volume means the rate of processor utilization, the number of messages, and the number of reductions or

suspensions that correspond to the type of profiling information. It's also possible to display only selected "What" items instead of all of them.

The bottom-left window in Fig. 2 is a When × Where view. The x axis is the cycle number, and the y axis is the processor number. This view displays the execution behavior of all goals on a 32-processor machine. The color patterns indicate the number of reductions. The relationship between the number of reductions and color pattern is displayed on the bottom right corner.

From this, we know that the work load on each processor was well balanced, and this program was executed about 70 000 reductions/an interval on each processor at each moment in time.

3.2.3 A What × Where view

There are two kinds of views in terms of "What" and "Where" items. One is a What × Where view which shows the load balance of each "What" item on each processor. A bar chart is displayed of a "What" item in order of total volume. The x axis represents the processor numbers, the y axis represents the rate of processor utilization, the number of messages, and the number of reductions or suspensions that correspond to the type of the profiling information. All bar charts are drawn with the same scale on the vertical axis, so it is easy to compare with the volume of each "What" item.

The other is an overall What × Where view which shows the load balances of all "What" items on each processor. Each "What" item is stacked in the same bar chart and displayed by a certain color pattern. The y axis represents the average rate of processor utilization, the total number of messages, and the number of total reductions or suspensions that correspond to the type of profiling information. The relationship between each category and color pattern is displayed on the top-right corner.

The top-right window in Fig. 2 shows the low-level behavior of the processor with an overall What × Where view. In this window, each categories of low-level behavior is displayed with several color pattern.

From this, the average of computation took more than 80 % of total execution time, and the average of communication on processor No. 0 was about 10 %, and the others were less than 5%. Since processor No. 0 collected answer values from the others, it took higher average. Thus, this view shows most of the processors run fully, and this example program was executed very efficiently on each processor.

4. Examples

This chapter discusses which views to use to view various performance bottlenecks. For efficient program execution on multiprocessor systems, the following phases are usually repeated until a solution is reached:

1) a program is partitioned into subtasks,
2) the subtask is mapped to each processor dynamically, and
3) each processor runs subtasks while communicating with each other.

Various problems are often encountered when executing a program on multiprocessor systems. We will show how graphic displays in both the higher program and lower implementation levels are helpful with performance problems.

4.1 Uneven partitioning

When the granularity between subtasks is very different, it is useful to observe the low-level processor behavior with a When × Where view and the higher-level processor behavior with a What × Where view. From the When × Where view, we will find which processors run fully and which are idle. From the What × Where view, we will determine which goals caused the load imbalances.

The left window in Fig. 3 shows the low-level behaviors on each processor with a When × Where view, while the right window in Fig. 3 shows the higher-level behaviors of the same processors with a What × Where view on a 21-processor machine. An example program is a logic design expert system which generates a circuit based on a behavior specification. The strategy of parallel execution is that first, the system divides a behavior specification into sub-specifications, next designs subcircuits based

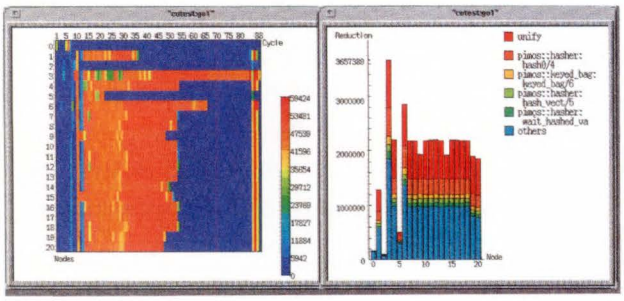


Fig. 3 – The low-level processor behavior (left) and execution behavior of goals (right).

on the sub-specifications on each processor, and finally gathers partial results together and combines them. The When × Where view suggests that most of processors run almost equally, but processors No. 3 and No. 6 run fully, and processors No. 0, No. 2, and No. 5 were idle. The What × Where indicates the which goals were executed on each processor.

From this, we know that processors No. 3 and No. 6 were allocated very complicated tasks, and processors No. 0, No. 2, and No. 5 were allocated very tiny tasks, that is, uneven partitioning of behavior specification must cause a bottleneck in performance.

4.2 Load imbalance

If a mapping algorithm has problems such as allocating subtasks to the same processor, it is useful to observe low-level behavior of the processor with a When × Where view and higher-level behavior with a What × Where view. From the When × Where view, we see which processors run fully or which are idle, and from the What × Where view, we see the load balance of each goal. Using both views, we can determine how to distribute the goals that are imbalanced to each processor.

The bottom-left window of Fig. 4 shows low-level behavior of the processor with a When × Where view, the top-left window and the top-right window show the higher-level behavior of the processor with an overall What × Where view, a What × Where view respectively. An example program is a part of the theorem prover which evaluates whether an input formula is a tautology. The strategy consists of 2

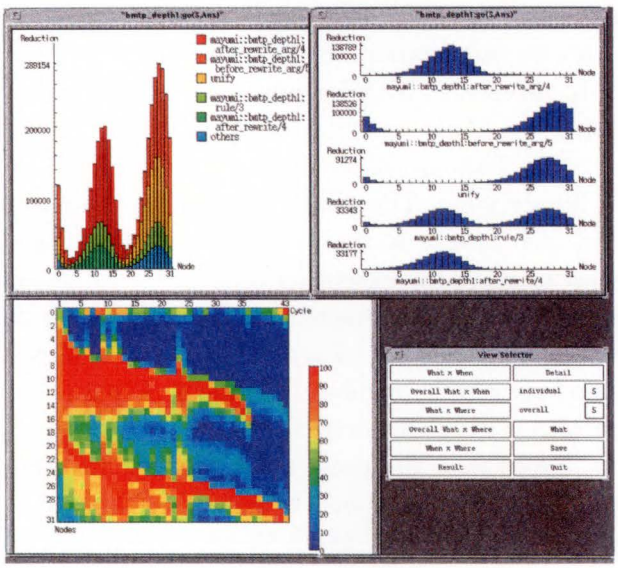


Fig. 4 – Low-level processor behavior (bottom-left), the load balances of all goals (top-left), and the load of each goal (top-right).

steps:

1) convert an input formula to clause form (i.e. conjunctive normal form),
2) evaluate its clause form and determine whether it is a tautology.

The step 1 is executed in parallel as follows. First, main task partitions an input formula into subformulas. Second, it generates subtasks to convert subclause forms, and finally, distributes subtasks to many processors dynamically. These steps are repeated recursively until subformulas are converted to subclause forms. The step 2 is executed in sequential on processor No. 0.

The When × Where view of the bottom-left window suggests that only certain processors (processor No. 6-15 and No. 23-31) run fully and that the others were mostly idle. The overall When × Where view of the top-left window also suggests most of the goals were executed on the same processors, especially the number of reductions of top five goals were higher than the other goals.

We can check the load of each goal on each processor from the What × Where view of the top-right window. These goals were executed on certain processors and were the cause of the load imbalances. From this, we have to change its mapping algorithm to be flatten the shape, to

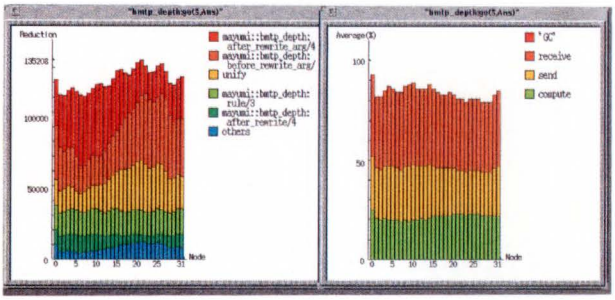


Fig. 5 – The load balances of goals (left) and low-level processor behavior (right).

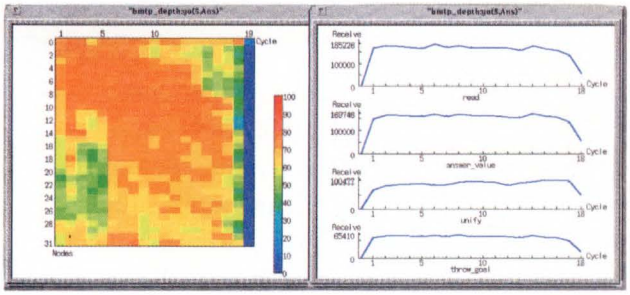


Fig. 6 – Low-level processor behavior about message handling (left) and message frequencies (right).

use all processors efficiently.

4.3 Large communication overhead

When subtasks are not mutually independent and must communicate with each other closely, the program is less efficient because of communication overhead. In this case, the low-level behavior of the processor with an overall What × Where view and frequencies of sending and receiving messages with a What × Where view are helpful. From the overall What × Where view, we will learn how much time has been consumed on message handling for each processor, while the What × Where view shows us what kind of messages each processor has sent or received.

Figure 5 displays an execution behavior of an improved version of the program described in section 4.2. The left window shows the load balances of all goals on a 32-processor machine with an overall What × When view. This view shows that the work load on each processor was balanced in overall execution, but was not efficient because of large communication overhead. It will be proved from low-level behavior of the processor with an overall What × Where view shown in the right window.

Figure 6 shows the same program execution as Fig. 5. The left window shows the receiving and sending message handling time rate with What × Where view, the right window shows the frequencies of four received inter-processor messages with a What × When view. The right window of Fig. 5 suggests the load average on each processor was about 80-85 %, but the average of computation on each processor was about 20 %. Most of the processing power was consumed sending and receiving message handling time more than 60 % of total execution time.

The left window of Fig. 6 shows the message handling time on each processor at each moment in time was almost equally. The right window in Fig. 6 shows that the read message was received about 185 000 times, answer_value message was about 170 000 times, unify message was 100 000 times, and throw_goal message was about 66 000 times per interval on all processors. The tasks generated in this program communicated with each other closely among processors as compared with the result of N queen's message frequencies (see the top-left window of Fig. 2).

From this, we know that as work loads are distributed more and more, it becomes easier to balance work loads on each processor, but communication overhead also increases and performance is thus lowered. As a result, we have to redesign or improve how to divide into subtasks. Because the generated subtasks that were not mutually independent caused such a problem we mentioned above.

5. Conclusion

We developed the ParaGraph system on parallel inference machines to provide graphic displays of processor utilization, interprocessor communication, and execution behavior of parallel programs. Experiments with various programs have indicated that graphic displays are helpful in dividing work loads evenly and determining where the bottlenecks are on multi-

processor systems.

We released a version last year as a tuning tool of PIMOS, but have experienced some problems. In the future, we will improve the system considering the following points. First, real-time performance visualization tools are needed. Although displaying execution behavior in real-time perturbs the program being monitored, it is useful not only in early tuning but also in debugging such as detecting deadlock status and infinite loops. To develop such a tool, low overhead instrumentation techniques and new displays that are easy to understand for programmers appearing in real-time must be devised.

Second, tools which can visualize the portion of the performance bottlenecks directly are needed. Massively parallel machines that have thousands of processors and programs for long runs produce a large amount of profiling information, but it is difficult to process or display for simple expansion of our system because of a vast quantity of information. To solve such problems, analysis techniques indicating bottlenecks directly will be needed. We will study automatic analysis techniques and graphical displays of its result (we call this *bottleneck visualization*). One such approach is critical path analysis[13], which identifies the path through the program that consumed the most time.

6. Acknowledgment

The work described in this paper was done under Institute for New Generation Computer Technology (ICOT) contract as a part of the R&D of the Fifth Generation Computer Systems Project. We thank all researchers of ICOT and other companies who tested our tool. We also thank K. Nakao and H. Kubo who helped us to develop this tool.

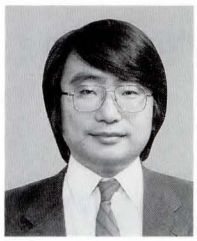
References

1) Ueda, K., and Chikayama, T.: Design of the Kernel Language for the Parallel Inference Machine. *The Computer Journal*, **33**, 6, pp. 494-500 (1990).
2) Goto, A., Sato, M., Nakajima, K., Taki, K., and Matsumoto, A.: Overview of the Parallel Inference Machine (PIM) Architecture. Proc. Fifth Generation Computer Systems 1988, **1**, Tokyo, pp. 208-229.
3) Chikayama, T., Sato, H., and Miyazaki, T.: Overview of the Parallel Inference Machine Operating System (PIMOS). Proc. Fifth Generation Computer Systems 1988, **1**, Tokyo, pp. 230-251.
4) Furuichi, M., Taki, K., and Ichiyoshi, N.: "A Multi-Level Load Balancing Scheme for OR-Parallel Exhaustive Search Program on the Multi-PSI". ICOT TR-526, Tokyo, ICOT Research Center, 1989.
5) Kimura, K., and Ichiyoshi, N.: Probabilistic Analysis of the Optimal Efficiency of the Multi-Level Dynamic Load Balancing Scheme. Proc. Sixth Distributed Memory Comput. Conf., 1989.
6) Scheifler, R. W., and Gettys, J.: The X Window system. *ACM Trans. Graphics*, **5**, 2, pp. 79-109 (1986).
7) Malony, A. D., Reed, D. A., and Rudolph, D. C.: "Integrating Performance Data Collection, Analysis, and Visualization". Performance Instrumentation and Visualization, 1st ed., N.Y., ACM Press, 1990, pp. 73-97.
8) Heath, M. T., and Etheridge, J. A.: Visualizing the Performance of Parallel Programs. *IEEE Software*, **8**, 5, pp. 29-39, (1991).
9) Aikawa, S., Kamiko, M., Kubo, H., Matsuzawa, F., and Chikayama, T.: ParaGraph: A Graphical Tuning Tool for Multiprocessor Systems. Proc. Fifth Generation Computer Systems 1992, **1**, Tokyo, pp. 286-293.
10) Ichiyoshi, N.: "Research Issues in Parallel Knowledge Information Processing". ICOT TM-0822, Tokyo, ICOT Research Center, 1989.
11) Nakajima, K., Inamura, Y., Ichiyoshi, N., Chikayama, T., and Nakashima, H.: Distributed Implementation of KL1 on the Multi-PSI/V2". Proc. Sixth Int. Conf. Logic Prgmg. 1989.
12) Nakajima, K., and Ichiyoshi, N.: "Evaluation of Inter-processor Communication in the KL1 Implementation on the Multi-PSI". ICOT TR-531, Tokyo, ICOT Research Center, 1990.
13) Miller, B. P., Clark, M., Hollingsworth, J., Kierstead, S., Lim, S., and Torzewski, T.: IPS-2:

The Second Generation of a Parallel Program Measurement System. *IEEE Trans. Par. Distr. Syst.*, **1**, 2, pp. 206-217 (1990).

Seiichi Aikawa received the B.S. degree in electronics from Gunma University, Gunma, Japan, in 1985.
He joined Fujitsu Laboratories Ltd., Kawasaki, in 1985 and has been engaged in reseach and development of programming environment in the Fifth Generation Computer Systems project.
He is a member of Information Processing Society of Japan.

Mayumi Kamiko received the B.S. degree in Information Eng. from Shinshu University, Nagano, Japan, in 1987.
She joined Fujitsu Laboratories Ltd., Kawasaki, in 1987 and has been engaged in reseach and development of operating system and programming environment in the Fifth Generation Computer Systems project.
She is a member of Information Processing Society of Japan.



Takashi Chikayama received his bachelor degree in mathematical engineering in 1977 and his doctoral degree in information engineering in 1982, both from the University of Tokyo.
He joined Fujitsu Laboratories Ltd., and then Institute for New Generation Computer Technology (ICOT) in 1982.
He has been conducting research and development in the Fifth Generation Computer Systems project since, mainly in programing language and operating systems area.

UDC 519.682:681.326

A Practical Test Program Generator Based on Attributed Grammer

● Hiroshi Kawata ● Hiroshi Saijo ● Chikao Shioya

(Manuscript received December 1, 1992)

This paper presents a test program generator called TPGEN, which is based on attributed grammar. TPGEN generates a wide variety of test programs mainly for programming language processors. The generated test programs are executable and have self-checking code for validating execution results. The generated test programs are assured that they have specific testing covevage.

TPGEN simulates the execution of a test program being generated and if an abnormal event such as zero divide or infinite loop is detected, TPGEN back-tracks to the specified position and selects an alternative production rule to avoid such abnormal execution. Introduction of this mechanism has succeeded in generating a wide variety of programs with complex structures.

1. Introduction

In the past, the formal definition of programming languages has been of interest mainly for the automatic generation of language processors such as compilers, interpreters and syntax-directed editors[1]. There have also been studies on its application to automatic generation of test cases or test programs[2]. Automatic generation of test programs typically defines test grammar in a formal way, such as BNF, and generates test programs from this description. It is a relatively simple task to randomly generate test programs according to a syntax description of the language, but the generation of practical executable programs requires solutions to several problems.

The first problem is to resolve contextual dependencies when generating correct programs. Duncan[3] has resolved this problem using attributed grammar and has developed a test program generator using a parser generator technique. But in our experience, the use of a general parser generator technique requires the description of all the information regarding attributes and the passing of attributes and thus results in a large and unwieldly description.

The second problem is in the generation of test programs with self-checking code. Confirmation of test results requiring a large amount of manpower reduces the benefit of automatic generation. Reports[3)-5)] show how to describe the semantics of language elements and give predicted execution results, but include no mechanism for automatic checking of execution results. D. L. Bird and C. U. Munoz[6] have described the automatic generation of test programs which are as executable and self-checkable as possible, although there are some restrictions on generated program structures.

The third problem concerns functional coverage. To assure adequate coverage we must be able to generate executable programs with complex structures such as loops. The reports that have been published so far describe relatively simple cases[3)-8)].

Other reports[9), 10)] have pointed out that PROLOG is very useful when prototyping a test case or test data generator. We implemented TPGEN in LISP because of LISP's facilities such as manipulation of pointer variables and complex data structures which were necessary to make our tool more practical.

Our test program generator, TPGEN, has been in use for software product inspection for

more than three years. In this paper, we present how TPGEN generates executable test programs with self-checking code, how it assures testing coverage, and how it improves the quality of generated programs and some empirical results obtained in comparison with conventional methods.

2. Outline of generation principle of TPGEN

In a syntax-directed definition, each production rule $A -> \alpha$ has associated with it a set of semantic rules of the form $b := f(c_1, c_2, \ldots c_k)$, where f is a function, b is a synthesized attribute of A or an inherited attribute of one of the grammar symbols on the right side of the production, and c_1, c_2, ... c_k are attributes belonging to the grammar symbols of the production. Functions in semantic rules are often written as expressions. Occasionally, the only purpose of a semantic rule in a syntax-directed definition is to create a side-effect. Such semantic rules are written as procedure calls or program segments. They can be thought of as rules defining the values of dummy production[1]. An attribute grammar is a syntax-directed definition in which the functions in semantic rules cannot have a side-effect.

The semantic definition of TPGEN consists of two descriptions: one resolves context-dependency to generate grammatically correct programs and the other simulates execution of generated programs. In generating a proper expression, for example, its type is passed to the production rule of an expression as an inherited attribute. The value attribute is introduced to each non-terminal so that the generated program can be simulated. Introduction of such attributes is not enough to complete the semantic definition of TPGEN, which will be explained later.

We will now explain how TPGEN generates test programs from the language definition. Figure 1 shows a simplified definition of a small subset of FORTRAN (see the appendix for more details). In this figure, symbols enclosed by < and > mean non-terminals and symbols surrounded by " and " mean terminals.

If we select production rules in Fig. 1 in the order of (1), (2-2), (4), (6-1), (7-1), ..., then parts (a)

```
<program>     -> <stmt> &                                  ...(1)
<stmt>        -> <assign-stmt> !                           ...(2-1)
                 <if-stmt>     &                           ...(2-2)
<assign-stmt> -> <var> "=" <expr>
                 (insert-checking-routine) &              ...(3-1)
<if-stmt>     -> "IF (" <b-expr> ") THEN" <stmt>
                 "ELSE" <stmt> "ENDIF"     &              ...(4)
<expr>        -> <primary>                 !              ...(5-1)
                 <primary> "+" <primary>   !              ...(5-2)
                 <primary> "-" <primary>   &              ...(5-3)
<b-expr>      -> <primary> ".GT." <primary> !              ...(6-1)
                 <primary> ".LT." <primary> !             ...(6-2)
                 <primary> ".EQ." <primary> &             ...(6-3)
<primary>     -> <ref-var>  !                             ...(7-1)
                 <const>    &                             ...(7-2)
```

Fig. 1 – A simplified example of language definition (syntax only).

```
IJK1 = 150                                                ...(c)

IF (IJK1 .GT. 100) THEN                                   ...(a)
  IJK1 = IJK1 - 60                                        ...(a)
  CALL  CHECK(1,90,IJK1,'ASSIGN STMT INVALID')...(b)
ELSE                                                      ...(a)
  IJK2 = 30                                               ...(a)
  CALL  CHECK(2,30,IJK2,'ASSIGN STMT INVALID')...(b)
ENDIF                                                     ...(a)
STOP
END
```

a) Example (1)

```
IJK1 = 150                                                ...(c)

IF (IJK1 .GT. 100) THEN                                   ...(a)
  IJK1 = IJK1 - 60                                        ...(a)
ELSE                                                      ...(a)
  IJK2 = 30                                               ...(a)
ENDIF                                                     ...(a)
CALL  CHECK(1,90,IJK1,'ASSIGN STMT INVALID')  ...(b)
STOP
END
```

b) Example (2)

Fig. 2 – Examples of generated text.

and (b) of Fig. 2a) will be generated. Part (b) of Fig. 2a) is generated based on the description of 'insert-checking-routine', and part (c) is an initialization statement which is generated with a declarative statement.

The procedure adopted by TPGEN to generate executable test programs with self-checking code is as follows:

1) TPGEN selects production rules randomly or considering functional coverage, if specified, starting from <program> and

generates source text. Usually, we define a <program> so that it includes several executable statements with several declarative statements and initialization statements. Number of statements included in a <program> is determined randomly within specified minimum and maximum integers.

2) TPGEN generates source text based on the selected production rules, and each time a production rule is applied, generated text is simulated. If an abnormal event such as an overflow is detected, some alternative is selected. If all alternatives result in abnormal execution, TPGEN back-tracks to the parent production rule of the current production rule and continues processing. Confirmation of execution results is done by generating self-checking code according to the 'insert-checking-routine'.

3) A source text for a <program> is generated by the above procedure. If a generated program includes a complex program structure such as a loop, there are several problems to be resolved, which will be explained later.

2.1 Resolving context-dependency

If a test program is generated by selecting production rules completely at random, variables or functions defined in the declaration portion will not coincide with those used in the execution portion. In order to resolve such context-dependency, information concerning declared variables must be easily retrieved. In TPGEN, system functions are available which make it easy to store and retrieve information concerning declared variables.

Such information is considered to belong to a specific non-terminal ($PROGRAM in the appendix). For example, if a variable is declared in a declarative statement, its name, data type and other information is registered to that non-terminal using a system function. If a variable is assigned a value by an assignment statement, the value of that variable is updated using another system function.

In generating an expression, the specific data type is passed to the production rule of an expression as an inherited attribute. In generating a subscript expression, its range or expected value is passed to the rule of an expression and if the value of the generated expression is not appropriate, we usually specify generation for a fixed period of time until it is appropriate. A back-tracking mechanism is very useful in such a situation. This will be explained later.

The production rule of subroutines is invoked from the semantic definition of the "CALL" statement, receives the necessary information (subroutine name and parameters) from it, and generates an appropriate subroutine. Normal execution of that subroutine is assured for the current "CALL" statement by simulating its execution at the time of generation.

The production rule of the "CALL" statement includes two alternatives. One is to generate a "CALL" statement for already generated subroutines, and the other is for a new subroutine. When a new subroutine is generated, to generate it must be stored with its name in a global variable so that other "CALL" statements for it can be generated later.

The generation of a subroutine at the time of generating a "CALL" statement, however, requires placing that subroutine at an appropriate point inside the generated test program. This problem is resolved by separating text generation and its arrangement. Syntax definition of such a production rule simply states the arrangement of generated text (syntax elements), and the generation of text is done by semantic definition (see the appendix).

2.2 Self-checking code

Automatic checking of execution results is very important in the inspection and testing of our software. We have been using checking routines for many years, before TPGEN was introduced. We have checking routines for each type of variable and for each target language. The checking routines themselves are coded in each target language. In Fig. 2, they receive, as parameters, a sequential number to identify erroneous text, a simulated value of the variable to be checked, the variable to be checked, and

error message text.

A test program generated by TPGEN is executable once it is link-edited with the above checking routines.

Although TPGEN understands values of all variables, users of TPGEN must specify the position where the self-checking code should be inserted, and the way it should be inserted. One reason is that TPGEN does not understand the structure of the target language. For example, if the "THEN" clause of an "IF" statement consists of one assignment statement, insertion of the self-checking code inside the "THEN" clause may require, in some languages, grouping of these statements. But the current TPGEN does not understand the target language to that extent.

Insertion of a self-checking routine is done for specified variables based on the definition of the 'insert-checking-routine' as shown in Fig. 1 or CHECK, which is described in the appendix. Figure 1 specifies the self-checking code to be inserted at the end of each assignment statement. In Fig. 2a), such code is inserted in the "THEN" clause and "ELSE" clauses. The "ELSE" clause is not executed in this example, but such an insertion is done assuming the "ELSE" clause will be executed. Another test program designer may specify the insertion of checking routines at the end of the "IF" statement for all variables whose values are changed during the execution of the "IF" statement. In this case, he should know the variables whose values change according to the difference between the value of variables on entrance to the "IF" statement and the value of variables on exit from the "IF" statement and a program described in Fig. 2b) is generated.

If the assignment statements in Fig. 2a) are included in a loop, the definition of CHECK in the appendix is not enough to generate a correct self-checking code. Insertion of a self-checking code inside a loop requires to identify repetition in addition to the value of the variable at that repetition and this information must be included in the definition of 'insert-checking-routine'.

Validation of the contents of external files is done by validation of variables when they retrieve a record from that file.

Test programs generated by TPGEN thus have self-checking code, and if a test program is executed correctly, such a program is discarded and only the information concerning what kind of functional test was done is stored in the database.

3. Characteristics of TPGEN

TPGEN generates test programs as described above. However, we also added the following features in order to make the quality of generated test programs closer to that of those generated manually.

3.1 Preventing abnormal execution by back-tracking

In TPGEN, an expression is evaluated each time a production rule is applied, and if an abnormal event is detected, TPGEN randomly

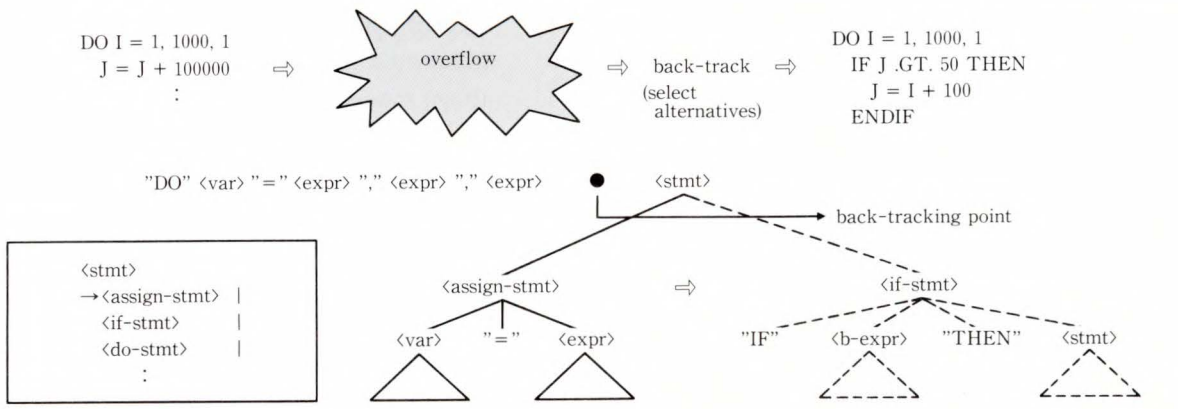


Fig. 3 – An example of back-tracking caused by a loop.

selects an alternative rule, excluding already selected rules. If an abnormal event is detected for all such selections, TPGEN back-tracks to its parent production rule, and text generation and its simulation continue. But this approach has some problems. One problem is that even if some selection, for example, (5-2) of Fig. 1 results in abnormal execution, it may be executed normally if it is selected more than twice, because a different <primary> will be selected.

A similar problem exists when such an expression is included inside a loop. In a loop (see Fig. 3), an assignment statement is generated so that no abnormal execution will occur for the first repetition. But the "DO" statement may cause abnormal execution for that assignment statement, at a later repetition. If it does, TPGEN considers that the "DO" statement, not the assignment statement, is executed abnormally. So, if the back-tracking point is specified inside the production rule of the "DO" statement, as in Fig. 3, the generation environment is resumed to the specified point, and the "DO" statement (body of the "DO" statement) is generated and simulated for a fixed number of times until it is executed normally. If a back-tracking point is specified at the top of the "DO" statement, generation of the "DO" statement itself is repeated for a fixed number of times until it is executed normally. If no back-tracking point is specified for the "DO" statement, some statement other than the "DO" statement will be selected as an alternative to the current "DO" statement.

To specify back-tracking points is delicate work. Users must make the scope of back-tracking as narrow as possible so that a wide variety of programs will be generated, and they must at the same time, reduce the frequency of back-tracking to improve generation efficiency.

Detection of infinite loops is done by counting repetition numbers. Since the introduction of "GOTO" statements makes it difficult to design test programs with no infinite loops, we usually design test programs which include "GOTO" statements separately.

3.2 Assuring functional coverage of generated programs

The combinations of selecting production rules can become enormous, even infinite, because of the nested or recursive structure of the target language. Thus, generation of test programs based on a random selection of production rules cannot answer such questions as; "are the generated test programs enough to cover the functionality of the target language?".

In addition to random selection of production rules, TPGEN tries to assure the following coverage of generated programs:

Condition-2 (2 level combination): For each alternative of each production rule, TPGEN tries to assure generation of all combinations of all alternatives of non-terminals which are included in that rule. For example, consider the <if-stmt> in Fig. 1. Each <b-expr> and <stmt> of the "THEN" clause, and the <stmt> of the "ELSE" clause consists of three alternatives, so twenty seven combinations should be selected for <if-stmt>.

This metric is based on syntax definition only, and it is usually not possible to generate test programs so that they satisfy condition-2 for all production rules. We adopted condition-2 for the following reasons:

1) As we cannot do complete functional testing, the second best approach is to make clear what kinds of functional tests are done by the generated programs.
2) Condition-2 above is, in a sense, close to the method which is actually used in designing test cases manually[11), 12)]. Thus, we can expect that the quality of test programs generated by TPGEN is close to that of those made manually.

3.3 Other features

The following additional features have been introduced to make TPGEN more practical.

1) Weights

A facility to control weights or relative frequency of each of the possible alternatives is introduced in the reports[5), 6)]. If one particular type of statement has a high weight, it will appear densely in the generated text. In TPGEN,

weights are introduced in the following way:

<stmt> -> W1 <assign-stmt> |
W2 <if- stmt> &

where W1/ Σ Wi is the probability of selecting <assign-stmt>.

Here, W1 and W2, may be expressions, and may be changed dynamically:

<stmt> -> 200 <assign-stmt> |
(E : SELECT : INIT 100 : IF-SELECTED (-30)) <if-stmt> &

The weight of <if-stmt> is read as follows: the relative frequency is set to 100 initially, then it is decreased by 30, each time <if-stmt> is selected. This enables us to change the selection frequency of <if-stmt> to zero at the time the nesting level of the IF statement reaches the maximum allowed by the target language processor.

2) Special terminals for formatting control

In FORTRAN, each line must start at column 7. We usually use indentation for nested IF statements. To cope with these matters, TPGEN has special terminals for controlling the position of generated text. This also improves the readability of generated text.

4. Evaluation of TPGEN

TPGEN has been used for more than 3 years in our quality assurance department for several language processors, including FORTRAN, C, LISP, PROLOG, AI-oriented shell, sort-merge, and COBOL-embedded SQL. At the time of their functional enhancement, these products were inspected partially, using TPGEN with about 2 800 production rules and more than 20 million LOCs (line of codes) of generated programs.

1) Applicable range of TPGEN

TPGEN is effective for generating test programs which execute normally. In the case of FORTRAN, about 80 % of the normal functional testing can be done using TPGEN. TPGEN is not effective for functional testing of special functions such as Γ function and special files such as VSAM files.

In the case of SQL, most of the functional testing for data manipulation language (DML) could actually be done using TPGEN, but TPGEN is not effective for data definition language (DDL). In the case of DML, the test program is designed based on a database whose structure is predetermined by a test case designer, but testing of DDL requires making a variety of databases, and it is difficult in our current environment to make simple few self-checking routines for such a varying data structure.

2) Quality of generated test programs

The functional coverage of TPGEN is basically the same as our conventional method, but we found that test programs generated by TPGEN have better bug-detection characteristics. The main reason, we think, is the complexity of generated programs. We analyzed test programs generated by TPGEN against those made by conventional methods for SQL test programs and found that the number of tokens included in a single SQL statement is about 3.5 times more than those made by conventional means. We also found that the number of phases, predicates, and the depth of nested expressions also increased.

In designing test cases manually, we often specify that some testing factor may be optional, because we think such a factor is not important for such a test case. But this is potentially a big problem, and bugs often exist in places where we think there are no problems. TPGEN generates test programs randomly without any preconceived ideas. This is the key point of a random testing tool.

3) Efforts required to make test programs using TPGEN

Our experience shows that making test programs using TPGEN is five times easier than conventional methods. Using TPGEN, most of our labors is devoted to designing test cases. The simple tedious work of coding the test programs is left to TPGEN, and our time can be spent on other work such as inspecting the ease of use and performance.

4) Performance

TPGEN requires a fair amount of CPU time and memory. It takes two or three seconds of CPU time on Fujitsu's large computer M-780 to generate test programs of about 1 kilo LOCs for a programming language which has no loops

```
      ....*....1....*....2....*....3....*....4....*....5....*....6....*....7....*
000001 $PROGRAM ->
000002   STMT   "@RNL"
000003   "STOP" "@RNL"
000004   "END"          #%
000005                  (E:EXP-R STMT :IN-TERM "@RNL") &
000006 STMT ->
000007   200 ASSIGN-STMT !
000008   100 IF-STMT     &
000009 ASSIGN-STMT ->
000010   VAR "=" EXPR "@RNL"
000011   CHECK                #%
000012                        (E:EXPL VAR)
000013                        (E:EXPL EXPR)
000014                        (M:SEM (F:VAR-SET VAR.V EXPR.V))
000015                        (E:EXPL CHECK VAR.V "ASSIGN STMT INVALID") &
000016 IF-STMT ->
000017   "IF (" B-EXPR ") THEN" "@TAB+"
000018      STMT#1              "@TAB-"
000019   "ELSE"                 "@TAB+"
000020      STMT#2              "@TAB-"
000021   "ENDIF"                #%
000022                        (E:EXPL B-EXPR)
000023                        (IF (NOT B-EXPR.V) (F:EFFECT-PART-CUT))
000024                        (E:EXPL STMT#1)
000025                        (IF B-EXPR.V (F:EFFECT-PART-CUT))
000026                        (E:EXPL STMT#2) &
000027 EXPR ->
000028   PRIMARY                                                        !
000029   PRIMARY#1 "+" PRIMARY#2 % (F:V-SET (+ PRIMARY#1.V PRIMARY#2.V)) !
000030   PRIMARY#1 "-" PRIMARY#2 % (F:V-SET (- PRIMARY#1.V PRIMARY#2.V)) &
000031 B-EXPR ->
000032   PRIMARY#1 ".GT." PRIMARY#2 % (F:V-SET (> PRIMARY#1.V PRIMARY#2.V)) !
000033   PRIMARY#1 ".LT." PRIMARY#2 % (F:V-SET (< PRIMARY#1.V PRIMARY#2.V)) !
000034   PRIMARY#1 ".EQ." PRIMARY#2 % (F:V-SET (= PRIMARY#1.V PRIMARY#2.V)) &
000035 PRIMARY ->
000036   REF-VAR % (F:V-SET (F:VARV REF-VAR)) !
000037   CONST                                &
000038 CHECK(VAR COMMENT) ->
000039   "CALL CHECK(" ITEM-NO "," A "," B "," C ")" #%
000040   (E:EXPL ITEM-NO)
000041   (E:PN A (F:VARV VAR))
000042   (E:PN B VAR)
000043   (E:PN C COMMENT) &
000044 VAR ->
000045   #P (CAR (F:RANDOM-SELECT (F:ALL-DECL-VAR)))      !
000046   #P (GENSYM "IJK") (F:DECL PO.V #'INTEGERP)       &
000047 REF-VAR -> #P (CAR (F:RANDOM-SELECT (F:ALL-VAR))) &
000048 CONST -> #P (F:RANDOM 1 200) &
000049 ITEM-NO ->
000050  A #%
000051  (IF (NOT ITN) (SETQ ITN 1))
000052  (E:PN A ITN)
000053  (SETQ ITN (1+ ITN)) &
000054 (SETQ *CVS* '(ITN))
```

Fig. 4 – Test program specification written in TPGEN for a small subset of FORTRAN.

such as SQL, and about 10 seconds for a programming language such as FORTRAN, where we need heavy testing of loops which often require back-tracking for selecting alternative production rules.

5. Conclusion

The test program generator TPGEN, which is based on attributed grammar, has succeeded in generating test programs which assure a specific testing coverage and have testing quality as good as or better than manually produced ones.

Additional merits of TPGEN in our practical work is important. In our quality assurance work, we sometime find that a software product has poor quality. We request the development group to take drastic measure to correct it. Later, when we receive the revised softwave product, we inspect it again. The same test set loses some capabilities for quality assurance in this case. With TPGEN, however,

we can generate another test set. Thus we can easily check the quality of the new product.

Testing using TPGEN is so-called 'black box' testing. We usually need to employ many kinds of tests including 'white box' testing, for software products testing. We have not evaluated TPGEN from the point of view of 'white box' testing. And simple definition of syntax and semantics of programming language is not enough as input to TPGEN, and the current TPGEN requires descriptions such as the insertion of checking routines and the specification of back-tracking positions.

Authors are grateful to Dr. Tokuda of Tokyo Institute of Technology for his helpful comments and suggestions on an earlier version of this paper.

6. Appendix

A detailed definition of a small subset of FORTRAN is shown in Fig. 4. The following is an explanation of this figure.

1) The data declaration and its related initialization are omitted.
2) The syntax definition is on the left side of "%" or "#%" and the semantic definition is on the right side of "%" or "#%". The numbers at the top of the syntax definition (see the definition of STMT) define the relative selection frequency of that rule (default is 100).
3) "@RNL" specifies the column where the generated text is placed. "@TAB+" and "@TAB−" indicate a carriage return and a shift of output position by a specified number of columns (default is 2) to the right or left, respectively.
4) In the case of "%", text is generated according to the syntax definition, and then the semantic definition is evaluated. The semantic definition of EXPR (line 29) means that the value attribute of the non-terminal EXPR should be set to the sum of the value attribute of PRIMARY#1 and that of PRIMARY#2. "F : V−SET" is a system function which evaluates the value of its argument and registers it as a value attribute of left side non-terminal. The "#*n*" is a sequential number to identify some non-terminal which appears more than twice in one syntax definition. If no semantic definition is described, as in line 28, the value attribute of the left side non-terminal is set to that of right side non-terminal.
5) In the case of "#%", the semantic definition is evaluated first, and then the text is arranged according to the syntax definition using text which is generated in the course of the evaluation of the semantic definition. Such semantic definition includes descriptions which control expansion of production rules. For example, semantic definition of $PROGRAM specifies to expand STMT randomly more than once (each STMT is separated by "@RNL").
6) In our actual implementation, semantic definition which specifies to simulate execution of the generated text must be explicitly stated by writing the "M : SEM" function as described in line 14, because such a definition may be executed more than once if the generated program includes a loop. But such description is omitted in this example except line 14.
7) Semantic definition of IF−STMT is a little bit complicated, because it contains a clause which is not executed. Simulation of such a not-executed clause is done in the same way as an executed clause, but the simulation environment (values of generated variables) must be resumed, on exit from such a not-executed clause, to those which is on entrance to the not-executed clause. Saving and restoring such simulation environment is specified by "F : EFFECT−PART−CUT". Line 23 means that if the value of B−EXPR is false, then save current simulation environment and restore simulation environment after having evaluated line 24.
8) In the definition of the assignment statement, the value attribute of the left side non-terminal (VAR) is name of some variable and the value attribute of expression (EXPR) is the value of that expression. On line 14, "F : VAR−SET" is a system function which retrieves the spe-

cified variable (VAR . V) among the information which is stored to $PROGRAM and sets its value field by the second argument (EXPR . V). Line 15 expands CHECK by passing two parameters (value attribute of VAR and string constant "ASSIGN STMT INVALID").

9) The definition of CHECK is read as follows: the CALL statement is expanded according to the syntax definition after evaluation of the semantic definition, which expands ITEM－NO (a sequential number that identifies the self-checking code), then A is set by the simulated value of VAR, B is set by VAR, and C is set by COMMENT.

References

1) Aho, A. V., Sethi, R., and Ullman, J. D.: Compilers-Principles, Techniques, and Tools. Addison-Wesley, 1986, pp. 279-289.
2) Ince, D. C.: The Automatic Generation of Test Data. *Computer Journal*, **30**, 1, pp. 63-69 (1987).
3) Duncan, A. G., and Hutchison, J. S.: Using Attributed Grammars to Test Designs and Implementation. Proc. 5 th ICSE, 1981, pp. 170-178.
4) Bauer, J. A., and Finger, A. B.: Test Plan Generation using Formal Grammars. Proc. 4th ICSE, Munich, 1979.
5) Mauer, P. M.: Generating Test Data with Enhanced Context-free Grammars. *IEEE Software*, 7, pp. 50-55 (1990).
6) Bird, D. L., and Munoz, C. U.: Automatic generation of random self-checking test cases. *IBM Syst. J.*, **22**, 3, pp. 229-245 (1983).
7) Bazzichi, F., and Spadafora, I.: An Automatic Generator for Compiler Testing. *IEEE Trans. Software Eng.* **SE-8**, 4, pp. 343-353 (1982).
8) Seaman, R. P.: Testing high level language compilers. Proc. IEEE Comput. Syst. and Tech. Conf., 1974, pp. 366-375.
9) Pesh, H., Schnupp, P., Schaller, H., and Spirk, A. P.: Test Case generation Using PROLOG. Proc. 8th ICSE, 1985, pp. 252-258.
10) Boug, L., Choquet, N., Fribourg, L., and Gaudel, M. C.: Application of PROLOG to Test Sets Generations. *Lect. Notes Comput. Sci.*, **186**, pp. 261-275 (1985).
11) Tatsumi, K.: Test Case Design Support System. Proc. Int. Conf. Quality Control (ICQC '87), JUSE, 1987, pp. 615-620.
12) Tatsumi, K.: Conceptual Support for Test Case Design. Proc. COMPSAC87, 1987, pp. 285-290.

Hiroshi Kawata received the B.E. degree in electrical engineering from Kyushu Institute of Technology, Fukuoka, Japan, in 1964 and Dr. degree in electrical engineering from the University of Tokyo, Tokyo, Japan, in 1970.
He joined Fujitsu Ltd., Kawasaki, in 1970 and has been engaged in development of compiler, database system, and various software tools. He is a member of Information Processing Society of Japan.

Hiroshi Saijo received the B.E. in physical engineering and the M.E. degree in tnformation engineerng from Tohoku University, Sendai, Japan, in 1977 and 1980.
He joined Fujitsu Ltd., Kawasaki, in 1980 and has been working on quality assurence of software products.
He is a member of Information Processing Society of Japan.

Chikao Shioya received the B.E. degree in electrical engineering from Keio University, Yokohama, Japan, in 1985.
He joined Fujitsu Ltd., in 1985 and has been working on quality assurence of software products. His Interest includes artificial intelligence.
He is a member of Information Processing Society of Japan.

UDC 621.3.011.7:681.32

Variable Ordering of Binary Decision Diagrams for Multi-Level Logic Minimization

● Masahiro Fujita ● Yusuke Matsunaga

(Manuscript received November 27, 1992)

Binary Decision Diagram (BDD) is now widely used in CAD fields, especially in formal verification and logic synthesis. In this paper, variable ordering methods of BDD for the application of multi-level logic minimization are presented. The variable ordering algorithm for sum-of-products representation is based on cover patterns and selects most binate variables first, and the one for multi-level logic representation is based on depth first traversal of circuits. In both cases, the obtained variable orderings are optimized by exchanging a variable with its neighbor in the ordering. Experimental results show the effectiveness of our methods.

1. Introduction

In logic synthesis, multi-level logic minimization plays a very important role in order to increase the quality of synthesized circuits in terms of area and testability. There have been many efforts in developing effective and efficient multi-level logic minimization methods, and several logic synthesis systems which include multi-level logic minimization have been developed [1)-6)]. In all of them, the key point of multi-level minimization is the use of don't care sets; i.e., people have been paying lots of attention to how to effectively use don't care sets and how to keep the size of don't care sets manageable. We have developed a multi-level logic minimization program[5)] based on the transduction method[4)] using Binary Decision Diagram (BDD)[7)] as an internal representation of logic functions. BDD is a canonical representation of logic functions. BDD has obtained much attention, since it can represent practical logic functions like the ones used in ALUs much more compactly than other representations, such as sum-of-products representation. Much larger circuits can be minimized using BDD compared with the original transduction method[4)] which uses truth tables to represent logic and permissible functions. We also developed a Boolean resubstitution algorithm with permissible functions[6)], which can be considered as an extension of the transduction method. Permissible functions are defined on each gate and express don't care sets which do not change the values of primary outputs. We used BDD to represent permissible functions compactly and get equal or superior performance compared with other multilevel logic minimization programs, such as MIS and BOLD, especially for large circuits.

The performance of our synthesis method, however, highly depends on sizes of BDDs. Sizes of BDDs greatly depend on the variable orderings used, especially for large circuits. In this paper, we present methods to find good variable orderings for BDDs with application to logic synthesis in mind. The problem of finding the best variable ordering is NP-hard[8)], and a couple of heuristics for *good* variable ordering were proposed [7), 9), 10)]. In Refs. 9 and 10 variable ordering methods based on network topology were developed. Here we use the approach that we first generate an initial variable ordering and then try to optimize it.

Initial variable orderings are generated in two different ways; if the synthesis program receives circuit descriptions in sum-of-products representation, the variable orderings are generated by analyzing cover patterns, and if the synthesis program receives circuit descriptions in multi-level logic representation, the variable orderings are generated by traversing the circuits in depth-first way as shown in Ref. 9. Both these situations can happen in logic synthesis. In some cases, specification for a circuit is in a truth table format, and in other cases, designers want to specify circuits with many intermediate variables which are actually multi-level logic representation. Sometimes designers want to optimize their circuit designs using multi-level logic minimization methods, in which case the input to logic synthesis systems is also in multi-level logic representation.

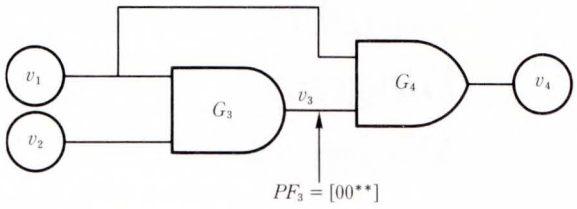
The initial orderings are optimized in the following way: First we construct BDDs for logic functions using the initial orderings, and then minimize sizes of BDDs by exchanging a variable with its neighbor in the ordering. The resulting orderings are used to calculate permissible functions for multi-level minimizations.

Since sizes of BDD highly depend on variable orderings, minimization time are also drastically influenced by the variable ordering used, although the quality of minimization results does not change. The required time for generation of initial orderings and optimization of them is much less than that for multi-level minimization. We present experimental results and show that we can get large speed-up by the presented methods.

In chapter 2, we briefly review permissible functions expressed in BDDs. In chapter 3, we present the method for initial ordering generation. In chapter 4, we present the method of BDD minimization after constructing BDDs for logic functions. Chapter 5 shows experimental results, and finally chapter 6 gives concluding remarks.

2. Boolean resubstitution with permissible functions and BDD

In this chapter, we briefly review the two key issues used in our multi-level logic minimization methods: permissible functions and BDD. As for the details, please see Refs. 5-7.



F_1	F_2	F_3	F_4
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

PF_3: A set of permissible functions

Fig. 1 – An example of permissible functions.

2.1 Permissible functions

The key concept of permissible functions is that each node in a circuit is an incompletely-specified logic function of the primary inputs due to the don't care sets obtained from network topologies, and permissible functions represent possible implementations at such nodes[4]. Permissible functions are defined on each node (a primary input, a gate, or a primary output) in the circuit. They are defined as follows. Assume v_i is an intermediate node in a network. The logic function of any output variable in the network may not change even when the logic function F_i of node v_i is replaced with another logic function PF_i. Then the logic function PF_i is called a permissible function of node v_i.

Usually, there is more than one permissible function for a node. Therefore, the don't care mark ($*$) is used to represent a set of permissible functions. Figure 1 shows an example of permissible functions. In this figure, v_1 and v_2 are input nodes, v_4 is an output node, and v_3 is an intermediate node. The F_i vector in the truth table represents a logic function of each node v_i. G_4 is an OR gate. Since the first and second values of F_4 are 0s, the first and second values of F_3 must remain to be 0s. The third and fourth values of F_4 are 1s and the third and fourth values of F_1 are 1s. Thus, the third and fourth values of F_3 may be either 0 or 1, and the logic function of F_4 does not change even when logic function F_3 is replaced with a logic function in

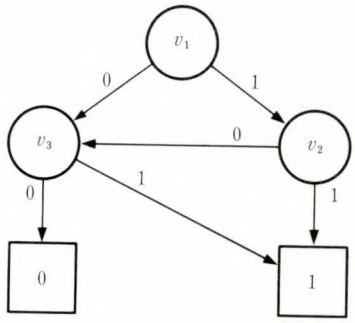


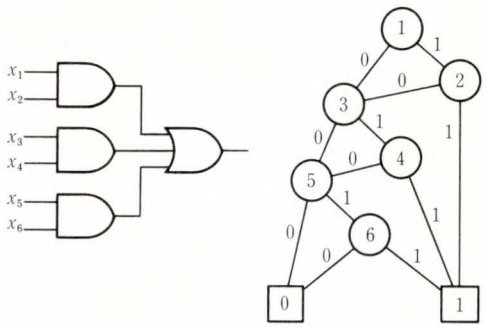
Fig. 2 – BDD representation of $F = v_1 \,\&\, v_2 + v_3$.

PF_3. Then PF_3 is the set of permissible functions of v_3. Though the logic functions and permissible functions in this figure are represented in terms of truth tables, in our implementation these are represented in BDD (see below). Permissible functions can be calculated by traversing networks from outputs to inputs. The details can be found in Refs. 4 and 5.

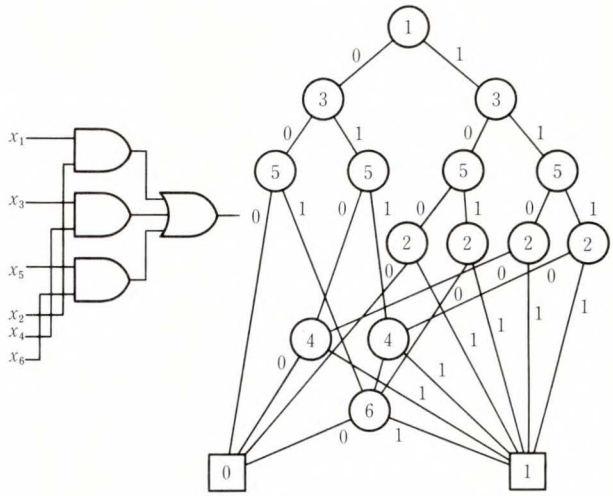
2.2 Binary decision diagrams

BDD or sometimes called Ordered BDD were proposed by Bryant[7]. A BDD is a kind of decision graph for representing logic functions with restrictions on the ordering of variables in the graph. Boolean functions are represented by directed, acyclic graphs with a vertex set containing two types of vertices. A non-terminal vertex has as attributes an input variable index and two children. A terminal vertex has as attributes a constant value 0 or 1 (to express permissible functions, we added one more constant '$*$' to express don't care value). Ordered means that if $x_i < x_j$ then all nodes with x_i precede all nodes with x_j. A path from the root to the terminal vertex with value 0 (or 1) gives a condition when logic function $f = 0$ (or $f = 1$).

Figure 2 shows an example of BDD representation of a logic function $F = v_1 \,\&\, v_2 + v_3$, where "&" represents AND and "+" represents OR. In this figure, a rectangle indicates a terminal node with a logical value, and a circle indicates a non-terminal node containing the variable index with the two children indicated by branches labeled 0 and 1. The variable ordering of this graph is $v_1 < v_2 < v_3$. Bryant developed efficient procedures for the operations



a) Best case



b) Worst case

Fig. 3 – Variable ordering.

of BDDs[7]. Those operations take time proportional to the sizes of BDDs.

Although BDD seems to be very promising, there is a big problem which must be resolved before we apply BDD to various areas. It is the variable ordering problem. The graph size heavily depends on the variable ordering. Figure 3 shows two different BDDs for the same logic functions using different variable ordering. In Fig. 3a), the best variable ordering is used, and in Fig. 3b), the worst variable ordering is used. As can be seen in the figure, as the number of 2-input AND gates increases (the number of input variables also increases proportionally), the size of the resulting BDD (the number of vertices in BDD) increases exponentially with the worst ordering, while these increase can be restricted to polynominal order if we use the best ordering. With a good ordering, BDD remains reasonably small for logic functions

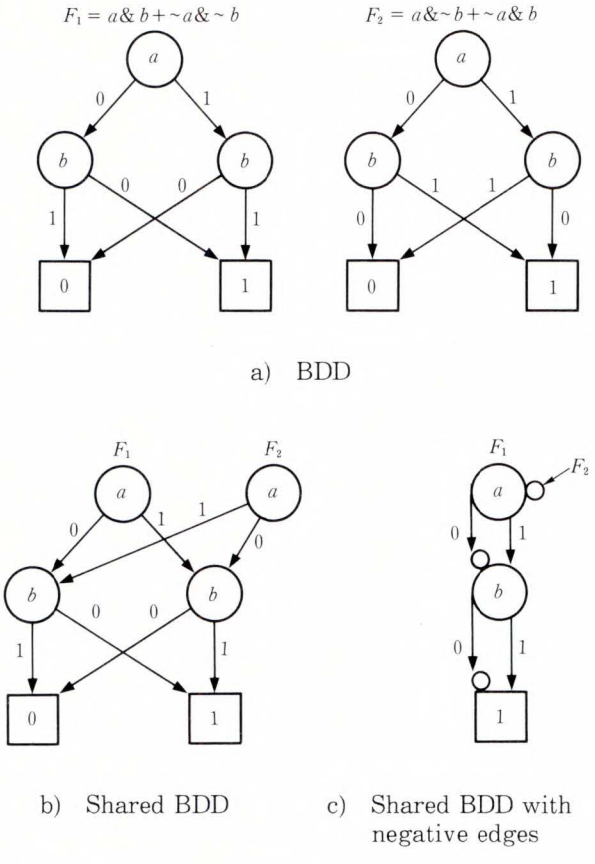


Fig. 4 – Shared BDD with negative edges.

which need exponential sizes in sum-of-products representation.

A graph can be shared with many logic functions and permissible functions, and the negative edge can be used to indicate an complemented logic[10]. This improvement enables the graph to be copied only by operating the pointer. The effective use of graph sharing and negative edges reduces CPU-time and memories significantly[6]. Figure. 4 shows an example of shared BDD with negative edges.

Although we use shared BDD with negative edges in real implementation, original BDD representation will be used in the following presentation for simplicity.

3. Generation of initial variable orderings

In the minimization process, BDDs for both logic functions and permissible functions are constructed and used. Generally, sizes of BDDs for logic functions are much (around 10 to 100 times) smaller than sizes of BDDs for permissible functions. Also, we can say by experiments that a variable ordering which gives smaller BDDs for logic functions also gives smaller BDDs for permissible functions. This means if we can get good variable orderings for BDDs of logic functions, we can also use them for BDDs of permissible functions effectively.

Here, we first construct BDDs for logic functions using the initial variable orderings generated by the heuristics, and then minimize sizes of BDDs by exchanging a variable with its neighbor in the ordering. In this chapter, we show the methods to generate initial variable orderings.

We have developed a variable ordering algorithm based on the heuristics: to minimize the number of net crossing when a circuit diagram is drawn, which is experimentally proved to be powerful[9]. This method is proven to be very effective for multi-level circuits by applying it to ISCAS test generation benchmark circuits.

Here we use two different methods for different circuit types; if the circuits are initially given in sum-of-product representation, we use the method which analyzes cover patterns, and most binate variables are ordered first. If the circuits are initially given in multi-level logic representation, we use the above heuristics[9].

The variable ordering method for circuits in sum-of-products representation is very simple; it first compute how binate each variable is in minimized cover expressions, i.e., how many complemented and uncomplemented variables appear in sum-of-products representation for circuit functions[1)-3)]. 0s and 1s appear in each column of the cover representation for circuit functions. So we first minimize a given sum-of-products representation by ESPRESSO[12] and get its minimized representation in cover format, which is matrix representation of sum-of-products representation[12]. We count up the number of 0s (which correspond to complemented variables) and 1s (which correspond to uncomplemented variables). Here 2s which correspond to don't care value in cover format are not counted up. Then each variable is ordered with its binateness, i.e., most binate variables are ordered

first. The most binate variable is the variable having the most number of 0s and 1s. If there is a tie, original ordering (appeared in the original benchmark circuit data) is used. Although this is very simple heuristic, it is proven to be very powerful by the experimental results shown in chapter 5.

4. Optimization of variable orderings

In this chapter we present a method to optimize variables orderings by exchanging a variable with its neighboring one. As we show in the following that we can easily get BDD for the variable ordering where only neighboring two variables are exchanged, if the BDD for the original variable ordering is given. This is because what we have to do is only to traverse and modify nodes relating to the two variables being exchanged.

Now suppose that we are exchanging the variable of i-th order with the variable of $(i+1)$-th order. Since BDDs are canonical forms, sub-BDDs having only nodes whose variable indices are from 1 to $(i-1)$-th and sub-BDDs having only nodes whose variable indices are from $(i+2)$-th to n-th remain unchanged even after the variables exchange. So, what we have to do is to modify parts of BDDs relating to the nodes whose variable indices are i-th or $(i+1)$-th.

There are several cases in the topology of those parts of BDDs, which are shown in Fig. 5. In this figure, f_1, f_2, f_3, f_4 represent different logic functions (or, in terms of BDDs, they point to different nodes). The first and simplest case is case 1 of Fig. 5; in the original BDD, only nodes whose variable index is $(i+1)$-th exist and there are no nodes whose variable index is i-th. In this case we only change the variable indices of the nodes, or in practical, there is no change in the BDD structure. The same situation hold for the case 2 of Fig. 5, where only nodes whose variable index is i-th exist.

The general case is shown in Fig. 5c). In this case, we change edges from the nodes of i-th and $(i+1)$-th variables as well as the variable indices of the nodes. However, if two of f_1, f_2, f_3, and f_4 are the same, we may eliminate some nodes, as shown in Fig. 5d). We can easily check it by

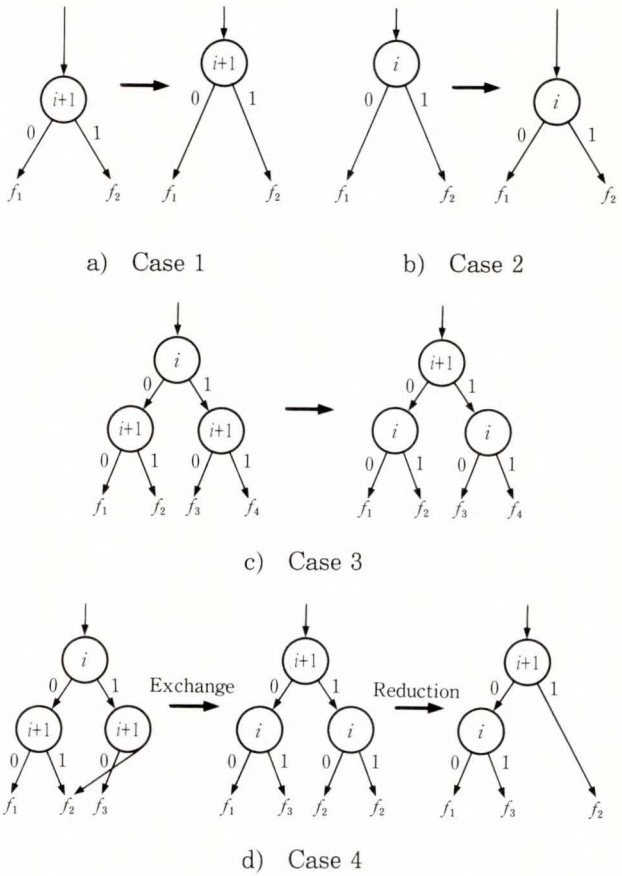


Fig. 5 – Exchange between i-th and $(i+1)$-th variables.

examining which ones are the same. After changing parts of BDDs as above, we execute the *reduce* operation in Ref. 7 only to those modified parts of BDDs. This procedure is the same for both BDDs and Shared BDDs.

A variable exchange example is shown in Fig. 6. In this figure, i-th and $(i+1)$-th variables are exchanged. We traverse from the root nodes, and when we first arrive in a node whose variable index is i-th or $(i+1)$-th (in the figure, nodes **A**, **B**, **C**, **D**, we apply the above procedure to modify parts of BDDs (when traversing, we first arrive the node **D** from the node **E** directly). For example, the node **A** is a case of Fig. 5d), the nodes **B** and **C** are reverse cases of Fig. 5d), and **D** is a case of Fig. 5a). So, the resulting BDD becomes the one as shown in the bottom of Fig. 6.

The above procedure is applied, and sizes of

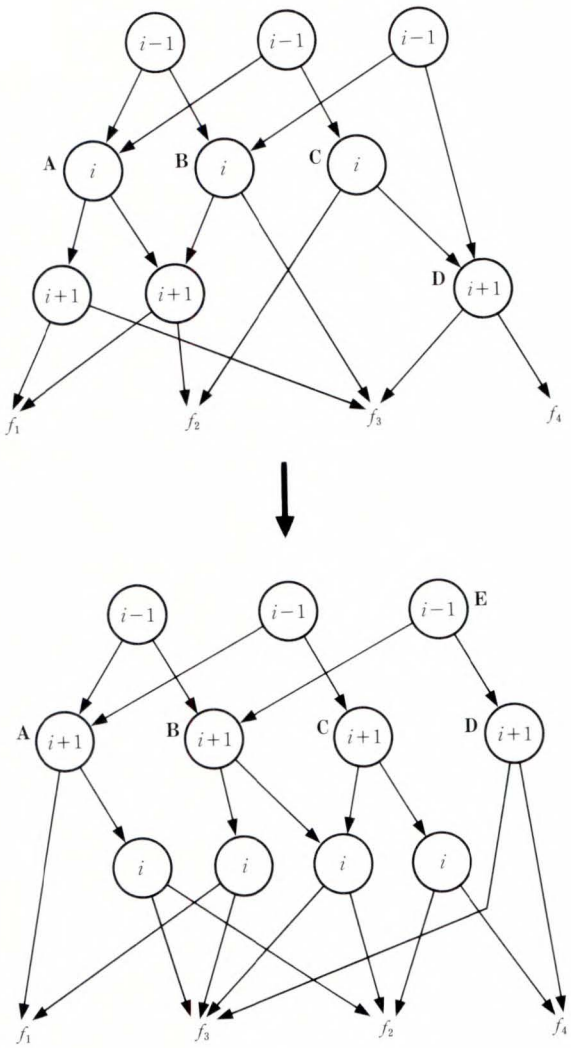


Fig. 6 – An example of ordering exchange.

BDDs before and after application of the procedure are compared. If there are some gains (the size of BDD after the application of the procedure is smaller), we really execute the variable exchange {we call here this procedure as Var_exchange (i)}. Var_exchange (i) is repeatedly applied by incrementing i until no improvement is made. We can say that if i-th and $(i+1)$-th variables can be exchanged, then in the resulting BDD, there is a possibility that those exchanged variables can be further exchanged with their neighbors. So, we mark those variable indices, and we apply Var_exchange (i) only to those marked ones. This control procedure is shown in Fig. 7.

```
Var_exchange_control()
{
 for each i {s[i] = 1}
 i = 1;
 while (some s[i] is 1) {
  if (s[i] == 1) {          /* only flaged index is tried */
   s[i] = 0;
   if (Var_exchange(i) == 1) { /* exchange index i with i+1, if gained */
    s[i-1] = 1;    /* maybe exchangeable */
    s[i+1] = 1;    /* maybe exchangeable */
   }
  }
  i = i + 1;
  if (i == n-1) i = 1; /* n is the number of primary inputs */
 }
}
```

Fig. 7 – Variable exchange sequence control procedure.

Table 1. Results of variable ordering methods applied to benchmark circuits in sum-of-products – Sizes of BDD for logic and permissible functions –

Circuits name	Nodes for logic functions			Nodes for permissible functions		
	Original	Heuristic	Exchange	Original	Heuristic	Exchange
Apex1	228 331	4 893	4 596	–	38 891	37 586
Apex2*	20 563	12 530	9 652	158 526	71 068	52 293
Apex3	–	5 621	5 635	–	40 956	41 163
Seq	258 595	7 424	5 869	–	39 665	32 386
Planet**	5 296	2 247	2 247	24 468	8 595	8 595
Sand	9 635	2 315	2 209	65 928	12 779	11 035
Styr	3 928	2 428	2 309	17 987	10 379	11 115
Scf	8 285	4 000	4 068	50 542	16 321	15 694

*: Subset of don't cares (similar to Ref. 2) are used in minimization.
**: There is no improvement by variable exchange.
Machine: SUN4/260

Table 2. Results of variable ordering methods applied to benchmark circuits in sum-of-products – Synthesis quality and time –

Circuits name	Final literals	Synthesis time (ratio: exchange = 1.0)		
		Original	Heuristic	Exchange
Apex1	1 139	– (-)	3 256 (1.04)	3 132 (1.0)
Apex2*	189	1 094 (4.01)	384 (1.41)	273 (1.0)
Apex3	1 117	– (-)	3 131 (1.01)	3 107 (1.0)
Seq	795	– (-)	5 089 (1.18)	4 318 (1.0)
Planet**	513	1 350 (3.64)	371 (1.00)	371 (1.0)
Sand	421	3 194 (7.46)	503 (1.18)	428 (1.0)
Styr	403	774 (2.03)	393 (1.03)	381 (1.0)
Scf	830	900 (3.03)	297 (1.00)	297 (1.0)

*: Subset of don't cares (similar to Ref. 2) are used in minimization.
**: There is no improvement by variable exchang.
Machine: SUN4/260
CPU time: second

Table 3. Results of variable ordering methods applied to benchmark circuits in multi-level logic

Circuits name	Nodes for logic functions			Final literals	Synthesis time (ratio)		
	Original	Heuristic	Exchange		Original	Heuristic	Exchange
Apex6	5 962	3 141	2 625	754	– (-)	341 (1.05)	326 (1.0)
Apex7	7 748	948	926	279	84 (2.21)	43 (1.13)	38 (1.0)
Rot	207 302	65 876	44 376	1 193	– (-)	– (-)	2 328 (1.0)
C432	11 262	11 262	8 387	194	72.2 (1.46)	72.2 (1.46)	49.6 (1.0)
C880*	–	27 656	24 763	413	– (-)	208 (1.06)	196 (1.0)
C2670*	–	129 587	91 434	853	– (-)	3 803 (1.35)	2 811 (1.0)
C5315*	–	162 232	73 962	2 061	– (-)	– (-)	7 451 (1.0)

*: Subset of don't cares (similar to Ref. 2) are used in minimization.
Machine: SUN4/260

5. Experimental results

We have applied the variable ordering methods presented earlier to logic synthesis benchmark circuits. Tables 1 and 2 show the results when applied to sum-of-products representation of the benchmark circuits. The second, third, and fourth columns in Table 1 show the number of nodes in BDDs for logic functions by the original variable ordering, the heuristic variable ordering presented in chapter 3, and the variable orderings after optimization of chapter 4, respectively. The fifth, sixth, and seventh columns show the number of nodes in BDDs for permissible functions in the similar way. We can see from the table that the variable ordering heuristic gives much better orderings than original orderings and those orderings can be further improved by the variable exchange method in chapter 4.

Tables 2 and 3 show the synthesis times for each variable ordering. The synthesis times in the columns of "exchange" include those for variable exchanges. Table 2 shows the results when applied to sum-of-products representation of the benchmark circuits whereas Table 3 shows the results when applied to multi-level representation of the benchmark circuits. Note that there is a strong correlation among the sizes of BDDs for logic and permissible functions and the synthesis times. The performance presented here is better than other synthesis tools in terms of synthesis speed and quality.

6. Conclusions

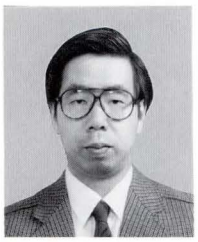
We have presented variable ordering methods of BDD. We used the approach that we first

generate an initial variable ordering and then optimize it. Initial variable orderings are generated by hueristics and the generated orderings are further optimized by exchanging variable orders. The experimental results by benchmark circuits show that our methods give very good ordering.

The presented method for optimizing variable ordering only exchange orderings for the neighboring two variables. It can be easily extended to change orderings for neighboring k-vartiables. If k is large, there is a chance to get much better orderings, although it can be very time consuming. There is a trade-off and it is one of the future research topics. Also, we plan to apply the methods to other application areas, such as sequential circuit verification and test generation.

References

1) Bartlett, K. A., Brayton, R. K., Hachtel, G. D., Jacoby, R. M., Morrison C. R., Rudell, R., Sangiovanni-Vincentelli, A. L., and Wang, A.: Multi-Level Logic Minimization Using Implicit Don't Cares. *IEEE Trans. Computer- Aided Design*, **CAD-7**, 6, pp. 723-740 (1988).
2) Saldanha, A., Wang, A. R., Brayton, R. K., and Sangiovanni-Vincentelli, A.: Multi-Level Logic Simplification using Don't Cares and Filters. Proc. 25th Design Automation Conf., 1989, pp. 277-282.
3) Bostick, D., Hachtel, G. D., Jacoby, R. M., Lightner, M. R., Moceyunas, P., Morrison, C. R., and Ravenscrofit, D.: The boulder optimal logic design system. Proc. IEEE Int. Conf. Comput. Aided Design'87, Santa Clara, 1987, pp. 62-65.
4) Muroga, S., Kambayashi, Y., Lai H. C., and Culliney, J. N.: The Transduction Method-Design of Logic Networks based on Permissible Functions. *IEEE Trans. Comput.*, **C-38**, 10, pp. 1404-1424 (1989).
5) Matsunaga, Y. and Fujita, M.: Multi-level Logic Optimization Using Binary Decision Diagrams. Proc. IEEE Int. Conf. Comput. Aided Design'89, Santa Clara, 1989, pp. 556-559.
6) Sato, H., Yasue, Y., Matsunaga, Y., and Fujita, M.: Boolean resubstitution with permissible functions and Binary Decision Diagrams. Proc. 27th ACM/IEEE Design Automation Conf., 1990, pp. 284-289.
7) Bryant, R. E.: Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.*, **C-35**, 8, pp. 667-691 (1986).
8) Friedman, S. J. and Supowit, K. J.: Finding the Optimal Variable Ordering for Binary Decision Diagrams. Proc. 24th ACM/IEEE Design Automation Conf., 1987, pp. 348-355.
9) Fujita, M., Fujisawa, H., and Kawato, N.: Evaluations and Improvements of a Boolean Comparison Method Based on Binary Decision Diagrams. Proc. IEEE Int. Conf. Comput. Aided Design'88, Santa Clara, 1988, pp. 2-5.
10) Malik, S., Wang, A. R., Brayton, R. K., and Sangiovanni-Vincentelli, A.: Logic Verfication using Binary Decision Diagrams in a Logic Synthesis Environment. Proc. IEEE Int. Conf. Comput. Aided Design'88, Santa Clara, 1988, pp. 6-9.
11) Minato, S. Ishiura, N., and Yajima, S.: Shared Binary Decision Diagram with Attributed Edges for Efficient Boolean Function Manipulation. Proc. 27th ACM/IEEE Design Automation Conf., 1990, pp. 52-57.
12) Brayton, R. K., Hachtel, G. D., McMullen, C., and Sangiovanni-Vincentelli, A.: Logic Minimization Algorithms for VLSI Synthesis. Kluwer Academic Publishers, 1984, 193p.



Masahiro Fujita received B.E. degree in Electrical Engineering, M.E. degree and Doctor degree in Information Engineering from the University of Tokyo, Tokyo, Japan, in 1980, 1982, and 1985, respectively.
Since he joined Fujitsu Laboratories Ltd. in 1985, he has been working on the research and development of the CAD for digital systems.
He is a member of Information Processing Society (IPS) of Japan.

Yusuke Matsunaga received B.E. and M.E. degrees in Electronics and Communication Engineering from Waseda University, Tokyo, Japan, in 1985 and 1987, respectively.
He joined Fujitsu Laboratories in Kawasaki, Japan, in 1987 and he has been involved in research and development of the CAD for digital systems. He is a member of Institute of Electoronics, Information and Communication Engineers (IEICE) and Information Processing Society (IPS) of Japan.

UDC 621.3.049.771.14

Performance and Hot Carrier Effects of Ultra-Thin-Film SOI/pMOSFET's at 90-300 K

● Kazuo Sukegawa ● Seiichiro Kawamura

(Manuscript received December 8, 1992)

This paper describes the characteristics and hot carrier effects of ultra-thin-film SOI/pMOSFET's between 90 and 295 K, and compares them with those of bulk MOSFET's. Decreasing the operating temperature further suppresses the short channel effects of ultra-thin-film SOI/pMOSFET's and improves their excellent current drivability. However, the positive threshold voltage shift caused by electrons that are trapped in the buried oxide during stressing is especially noticeable at low temperatures. Provided the supply voltage can be reduced, ultra-thin-film SOI/MOSFET's are promising candidates for deep-submicron MOSFET's operating at low temperatures.

1. Introduction

Compared with bulk MOSFET's, silicon-on-insulator (SOI) MOSFET's have no latch-up, a low parasitic capacitance, and enable a greater packing density. Moreover, ultra-thin-film SOI/MOSFET's have several additional advantages, for example, suppression of short channel effects, excellent subthreshold characteristics, and a greater carrier mobility[1), 2)]. These advantages make ultra-thin-film SOI/MOSFET's candidates for deep-submicron MOSFET's[3)]. Because scaled down MOSFET's require a reduction in supply voltage to maintain device reliability, the operation temperature must be reduced to maintain a sufficient on/off margin of gate voltage. In this respect, ultra-thin-film SOI/MOSFET's are especially attractive because they offer the above advantages even at low temperatures.

This paper compares the characteristics and hot carrier effects of ultra-thin-film SOI/pMOSFET's between 90 and 295 K with those of bulk pMOSFET's.

2. Experiments

2.1 Fully depleted SOI/MOSFET's

Unlike bulk MOSFET's and conventional SOI/MOSFET's, the Si film in the channel region of fully depleted SOI/MOSFET's is fully depleted under operational conditions, and the depth of the source/drain junction is equal to the Si film thickness (see Fig. 1). The advantages of ultra-thin-film SOI/MOSFET's mentioned in Chap. 1 are due to these features.

2.2 Device processing

In this study, separation by implanted oxygen (SIMOX) wafers with Si-films of 80-100 nm and a buried oxide layer of 520 nm were used. Single drain p-channel MOSFET's were fabricated on these wafers by the process described below. The active regions were defined by LOCOS isolation. Phosphorus ions were implanted into the channel region at energies between 30 and 40 keV at doses between 1.6×10^{11} and 2.0×10^{12} cm^{-2}. The 10 nm gate oxide was grown at 1 100 ℃ in O_2/Ar. An N^+ poly-Si gate electrode was patterned using RIE etching. BF_2 ions were implanted into the source/drain regions at energies between 35 and 50 keV at doses between 8.0×10^{14} and 1.0×10^{15} cm^{-2}. Implantation was followed by annealing for 20 minutes at 850 ℃ in N_2. Bulk MOSFET's were also fabricated on n-type, 10 Ωcm (100) Si wafers

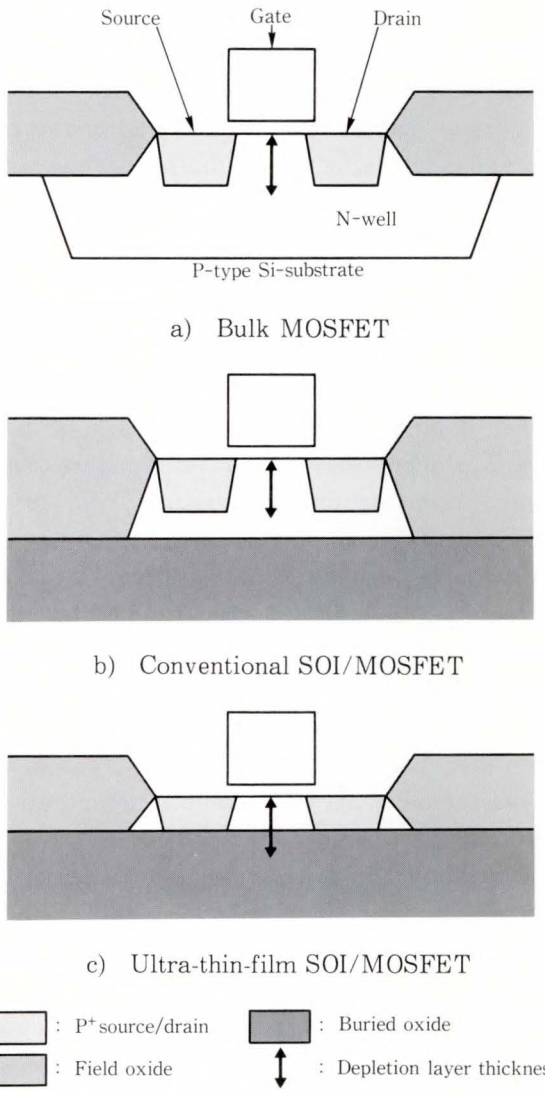


Fig. 1 – Cross sections of MOSFET's.

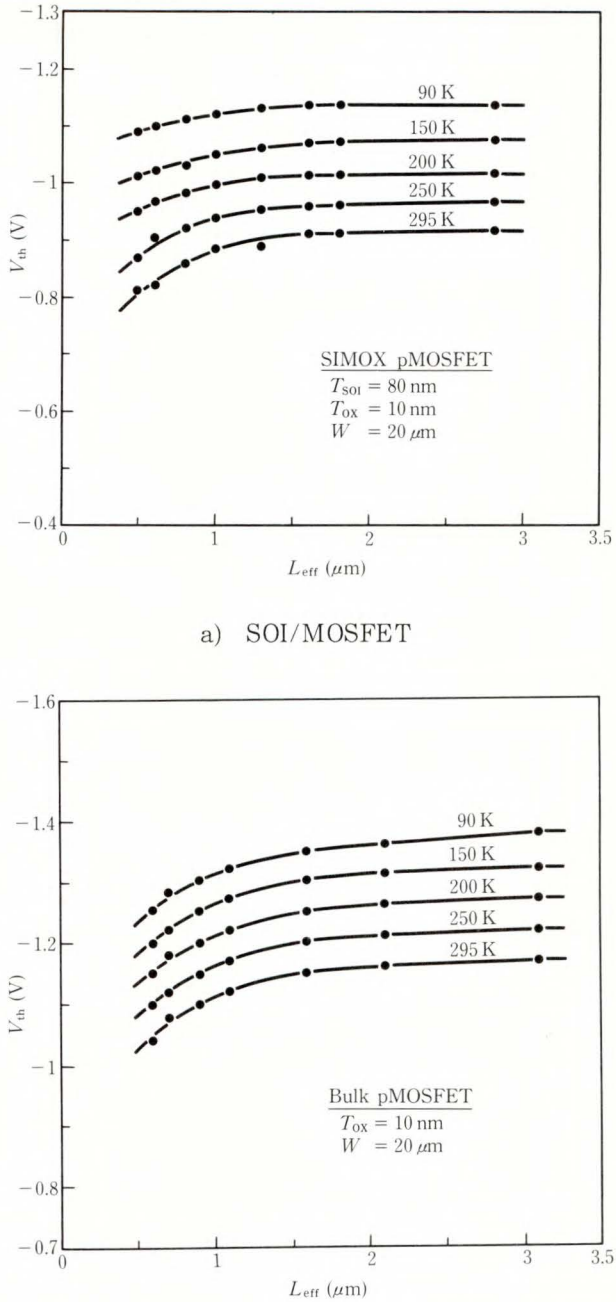
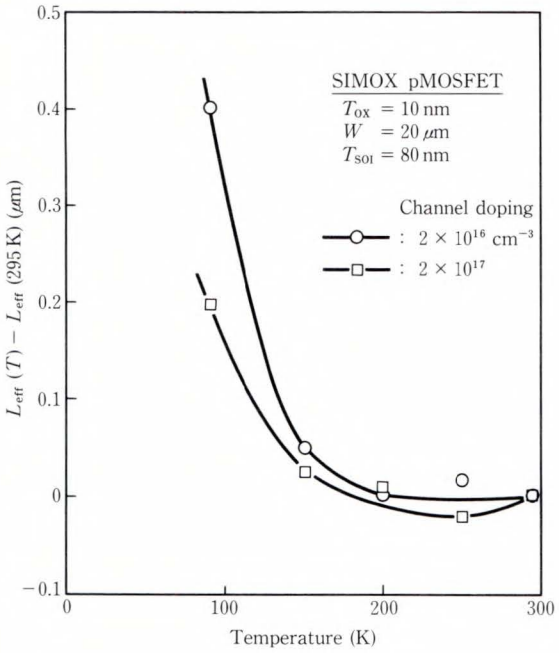


Fig. 2 – Threshold voltage versus L_{eff} for various temperatures.

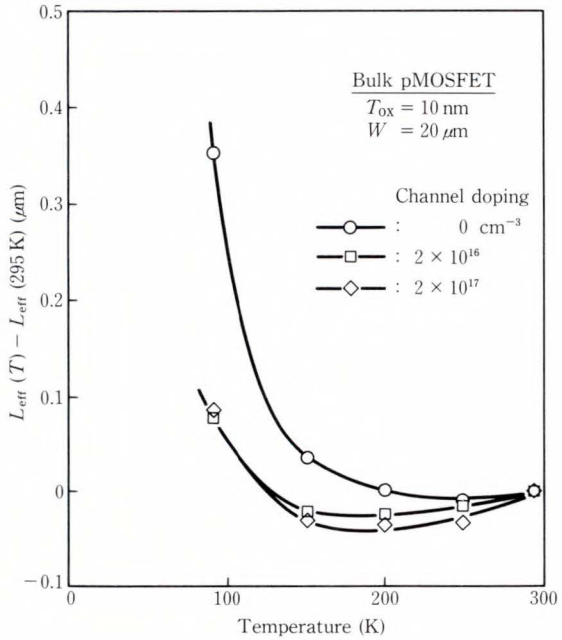
using the same process and channel control doping. The junction depth, X_j, of bulk devices was estimated by process simulation to be about 0.35 μm. The X_j of SOI devices is equal to the Si film thickness, and was 80-100 nm in this study.

2.3 Measurement

The effective channel length, L_{eff}, is obtained by using the Laux method[4]. In this study, devices with L_{eff} = 0.4-10 μm and a channel width of 20 μm were used. The characteristics of the SOI and bulk devices were measured at temperatures between 90 and 295 K. The characteristics in the linear region were measured at a drain voltage of −0.1 V and a back gate voltage of 0.0 V. In order to investigate the hot carrier effect, both devices were stressed at 90 K and 295 K with a drain voltage between −6.5 and −7.5 V and a back gate voltage of 0.0 V.



a) SOI/MOSFET



b) Bulk MOSFET

Fig. 3 – Temperature dependence of effective channel length.

3. Results and discussion

3.1 Short channel effects

Figure 2 shows how the threshold voltage, V_{th}, changes with L_{eff} for various temperatures for SOI and bulk devices having a channel doping of 2.0×10^{16} cm^{-3}. When L_{eff} is reduced, the V_{th} of both devices becomes more positive. Although short channel effects in SOI devices are greatly suppressed at low temperatures, short channel effects in bulk devices are independent of temperature. This difference can be explained by considering the temperature dependence of L_{eff} in SOI and bulk devices.

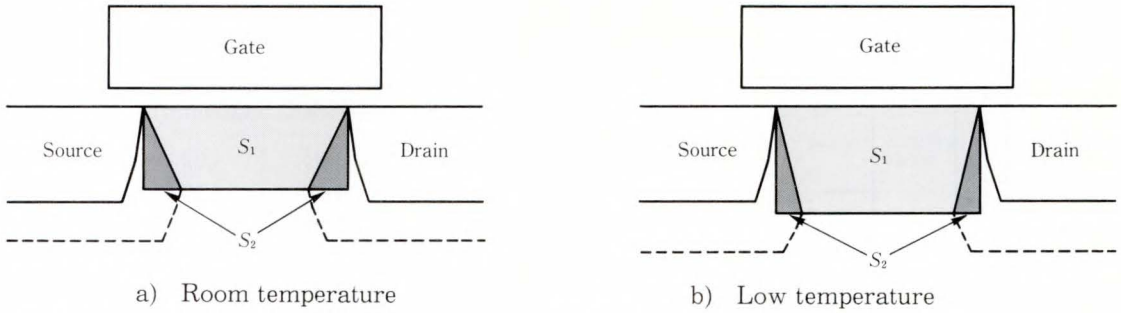
Figure 3 shows the differences between the L_{eff} at 295 K and the L_{eff} at temperatures between 90 and 295 K for various levels of channel doping. In low-doped devices of both types, L_{eff} increases at low temperatures. For example, reducing the temperature from 295 K to 90 K, increases L_{eff} by 0.40 μm in SOI devices with a channel doping of 2.0×10^{16}cm^{-3}, and by 0.08 μm in their bulk counterparts. On the other hand, short channel effects in bulk devices with a high channel doping are slightly dependent on temperature. These results show that in devices with a low channel doping, the gate controls the channel region more effectively at temperatures below 295 K.

It is known that low temperature operation suppresses short channel effects in bulk devices with a low channel doping[5]. As the temperature is reduced, the Fermi level of Si approaches the band edge (conduction band edge in this case). This increase in the Fermi level, ΔE_f, increases the depletion layer thickness, l_d, as follows (see Fig. 4):

$$l_d\,(\mathrm{MOS}) \propto (2 \cdot \Delta E_f)^{1/2},$$
$$l_d\,(\mathrm{p\text{-}n}) \propto (\Delta E_f)^{1/2},$$

These relations indicate that the channel region is effectively controlled by the gate at low temperatures. In devices with a lower channel doping, E_f is larger; therefore, the suppression of short channel effects is more pronounced.

We will now discuss the reasons for the suppression of short channel effects in SOI devices (see Fig. 5). S_1 and S_2 represent the depletion region controlled by the gate and the depletion region controlled by the source/drain, respectively. Points **A**, **B**, **C**, and **D** in the SOI/MOSFET shown in Fig. 5 are assumed to have the following characteristics:



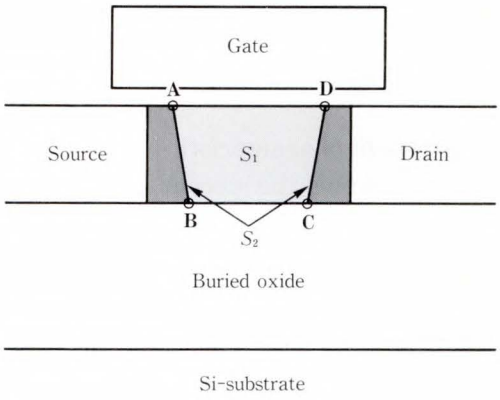
a) Room temperature

b) Low temperature

S_1: Depletion layer controlled by the gate

S_2: Depletion layer controlled by the source/drain

Fig. 4 – Charge sharing in bulk MOSFET.



Assumptions for points **A**, **B**, **C**, and **D**:
a) Lateral electric field is zero.
b) $E_f = E_i$.

Fig. 5 – Charge sharing model for SOI/MOSFET.

1) The lateral electric field is 0 V/cm.
2) The Fermi level is equal to the intrinsic Fermi level.

However, regardless of the correctness of these assumptions, the calculated charge sharing coefficients[6] for both devices $\{S_1/(S_1 + S_2)\}$ is independent of temperature. This implies that the two-dimensional charge distribution suppresses short channel effects in SOI devices.

Reducing the junction depth, X_j, further suppresses the short channel effects. The X_j for SOI devices (80-100 nm) is much smaller than that of bulk devices (350 nm). Therefore, at low temperatures, short channel effects are suppressed more effectively in SOI devices than in bulk devices.

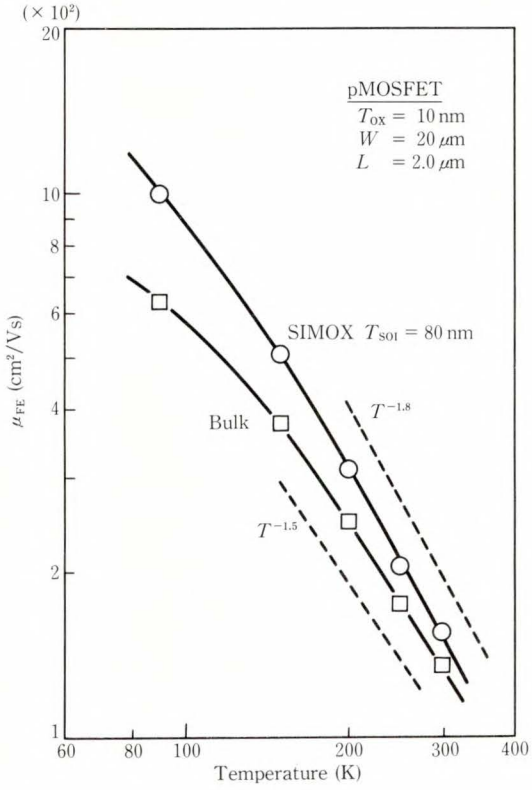


Fig. 6 – Temperature dependence of field effect mobility.

3.2 Carrier mobility

Figure 6 shows the dependence of field effect mobility, μ_{FE}, on temperature as determined from the transconductance, g_m, in the linear region of devices with a channel doping of 2.0×10^{16} cm^{-3}. The μ_{FE} of SOI devices is larger than that of the bulk devices throughout the temperature range. The μ_{FE} of SOI devices changes more noticeably with temperature than the μ_{FE} of bulk devices. The μ_{FE} of SOI devices

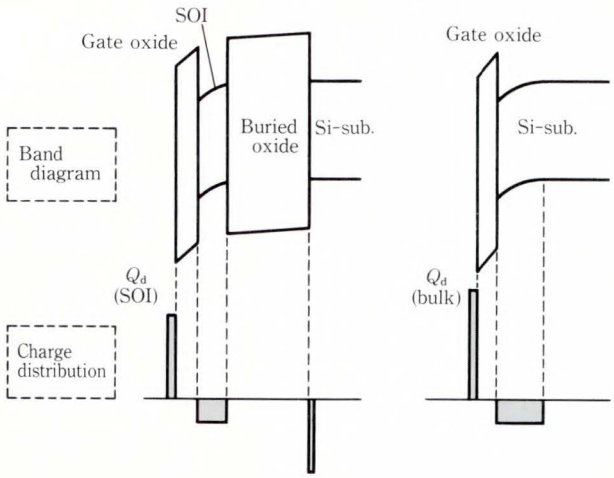


Fig. 7 – Band diagram and charge distribution for MOSFET.

increases from 155 to 1 000 cm^2/Vs over the range from 295 to 90 K. The change in bulk devices over the same range is from 130 to 630 cm^2/Vs. Assuming that μ_{FE} is proportional to T^{-x}, x is 1.8 for SOI devices and 1.5 for bulk devices, indicating that the μ_{FE} of SOI devices at low temperatures is larger than that of bulk devices.

Since both devices in the figures have the same channel doping, ionized dopant scattering affects the μ_{FE} of both devices to the same degree. Therefore, it can be assumed that the difference between the μ_{FE} of SOI and bulk devices is due solely to charges that form an inversion layer and establish an electric field, E_s, that is perpendicular to the channel direction. We will now discuss the effect of E_s on μ_{FE} based on calculations for a one-dimensional MOS structure (see Fig. 7). The total induced charge, Q_d, which forms the inversion layer is related to E_s by the following formula:

$$E_s = Q_d/\varepsilon_{OX}.$$

As E_s increases, μ_{FE} decreases[7]. The Q_d values for SOI and bulk devices at various temperatures are shown in Table 1. The table shows that:

1) At any temperature, the Q_d for SOI devices, Q_d(SOI) is less than that for bulk devices, Q_d(bulk).
2) The ratio Q_d(bulk)/Q_d(SOI) is inversely proportional to temperature.

Table 1. Total charge density required to form an inversion layer

T (K)	Q_d (SOI)	Q_d (bulk)	Q_d (bulk)/Q_d (SOI)
90	1.94	4.62	2.38
150	1.90	4.40	2.32
200	1.88	4.22	2.24
250	1.85	4.02	2.17
295	1.83	3.89	2.13

(unit) Q_d : $\times 10^{11}/cm^2$

The first of the above observations explains why the μ_{FE} of SOI devices is greater than that of bulk devices at any temperature. The second observation explains why this difference in μ_{FE} is enhanced at low temperatures.

3.3 Subthreshold characteristics

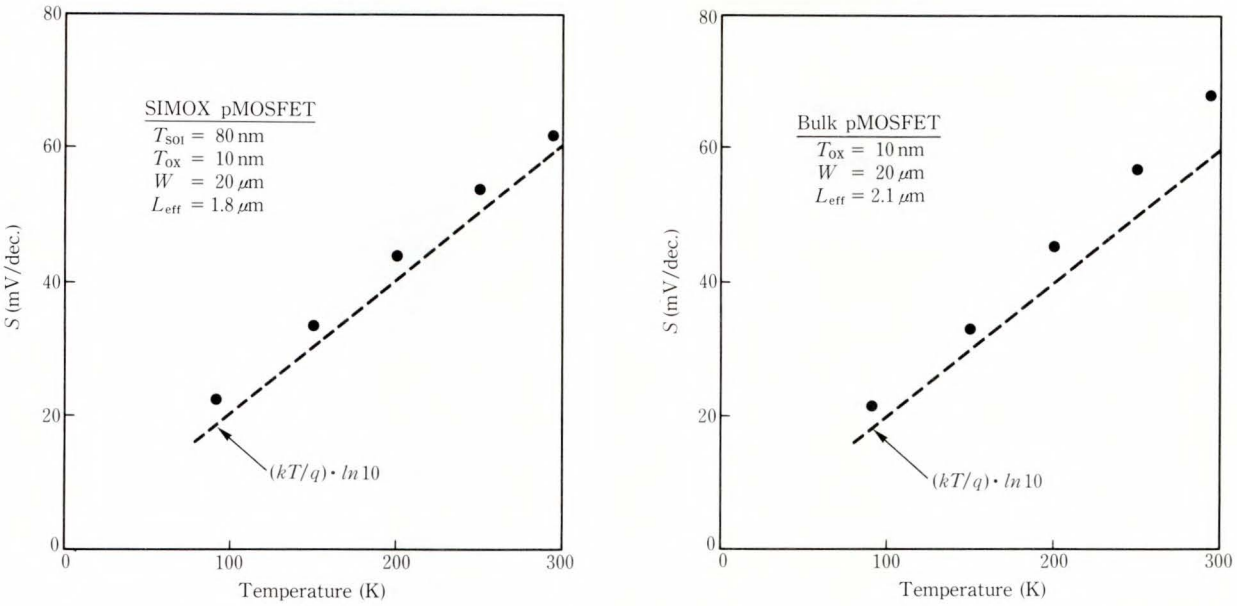
Figure 8 shows the dependence of subthreshold swing, S, on temperature for SOI and bulk devices. The dashed lines in the figure show the theoretical limits for S, i.e. $(kT/q) \cdot ln10$ at each temperature. The difference between the theoretical limits and experimental values for SOI devices is independent of temperature, but the difference for bulk devices decreases when the temperature decreases.

In general, S is given by the following equation[8]:

$$S = (kT/q) \cdot ln\ 10 \cdot \{1 + (C_d + C_{it})/C_{OX}\},$$

where C_d and C_{it} are the depletion layer capacitance and equivalent interface state capacitance, respectively. Since C_d for SOI devices is almost zero, S for SOI devices at 295 K is nearly equal to the theoretical value. However, the value of S for SOI devices may be partially determined by the effects of the interface states at the front gate (gate oxide/SOI) and the interface states at the back gate (SOI/buried oxide).

The ratios of the experimental values to the theoretical limits, i.e. $\{1+(C_d + C_{it})/C_{OX}\}$ in the above formula, are shown in Fig. 9. When the temperature decreases, the ratio for SOI rapidly increases, especially at 90 K. This can be attributed to the effects of the interface states at the back gate. The electrical characteristics of



a) SOI/MOSFET b) Bulk MOSFET

Fig. 8 – Temperature dependence of subthreshold swing.

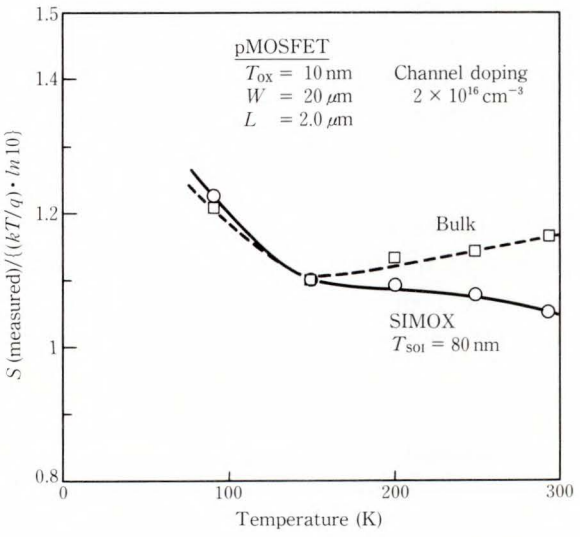


Fig. 9 – Temperature dependence of normalized subthreshold swing.

the buried oxide used in this study, which is formed by implanting O^+ into the Si substrate, seem to be inferior to those of the thermal oxide. When the temperature decreases, E_f and therefore the surface potentials at the front and back gates increase. Therefore, the increase in surface potential at the back gate interface at low temperatures may cause the subthreshold characteristics of SOI devices to deviate more rapidly from the theoretical than is the case for bulk devices.

3.4 Hot carrier effects

Hot carrier effects were investigated for both devices with an L_{eff} of 1.5 μm and a channel doping of 2.0×10^{17} cm^{-3}. Since the changes in the characteristics of bulk devices that are caused during stressing are most noticeable at the maximum gate current, I_g[9], we chose a stress condition for both devices such that I_g was maximum at $V_{ds} = -6.5$ V. The polarity of I_g shows that it is an electron current. This indicates that the electrons which are generated by impact ionization near the drain edge are injected into the gate oxide, and that some of them are trapped in the gate oxide.

Figure 10 shows how $\Delta g_m/g_{mo}$ changes with the stress time at 295 K and 90 K. For both devices, $\Delta g_m/g_{mo}$ is positive and is larger at 90 K than at 295 K. At both temperatures, the $\Delta g_m/g_{mo}$ values for SOI devices are larger than those for bulk devices. The $\Delta g_m/g_{mo}$ values are positive because of the reduction in L_{eff} due to fixed negative charges that are produced in the

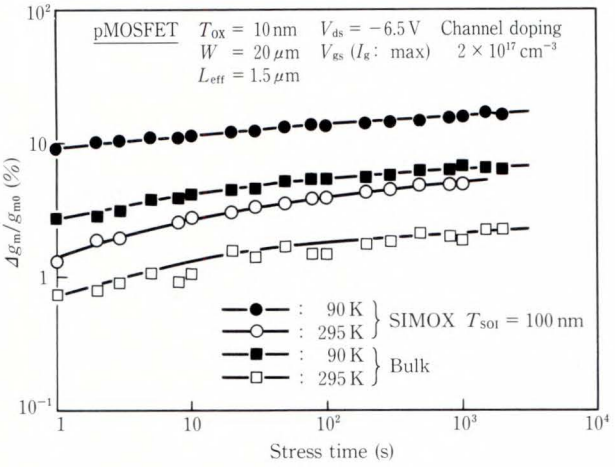


Fig. 10 – Change in transconductance versus stress time.

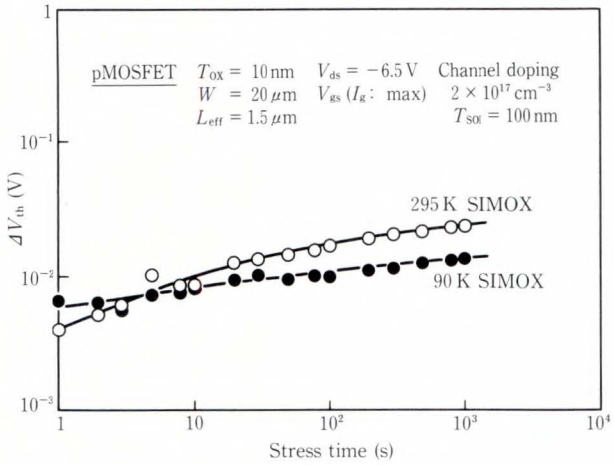


Fig. 11 – Change in threshold voltage versus stress time.

gate oxide by the electron injection performed during stressing. Analysis of the $\Delta g_m/g_{mo}$ of SOI devices that were stressed for 1 000s indicated that L_{eff} was reduced by between 0.10 and 0.15 μm.

Figure 11 shows how V_{th} changes with the stress time. Although ΔV_{th} can be detected only for SOI devices, this cannot be explained by the short channel effects caused by the reduction in L_{eff}.

The hot carrier effects in SOI devices are associated with the electrons injected during stressing and become trapped in the buried oxide. The effect of charges in the buried oxide

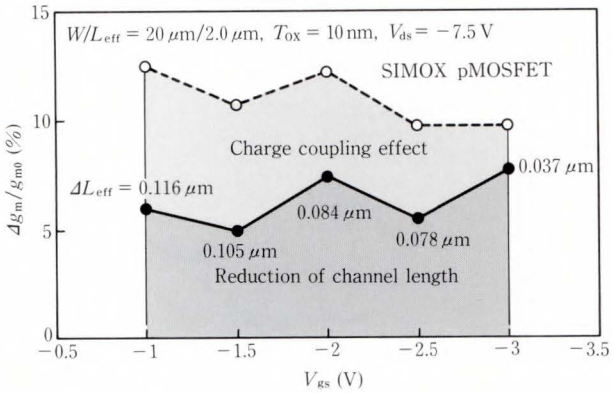


Fig. 12 – Effects of trapped electrons in gate and buried oxide on transconductance.

on the front gate characteristics is equivalent to the effect of substrate bias in bulk devices. Hereafter, this equivalent substrate bias effect will be called the charge coupling effect[10]. Trapped electrons cause a positive ΔV_{th} in SOI devices. The trapped electrons in the buried oxide increase the g_m of SOI devices because they reduce E_s[11]. Therefore, trapped electrons in both the gate oxide and the buried oxide affect $\Delta g_m/g_{mo}$. Figure 12 shows how the effects of the front gate and the buried oxide are separated by the method described in Ref. 11. The changes in SOI device characteristics occur not only because of the reduction of L_{eff} due to trapped electrons in the front gate, but also because of the charge coupling effect due to the electrons that are trapped in the buried oxide during stressing.

4. Conclusion

The characteristics and hot carrier effects of ultra-thin-film SOI/pMOSFET's at low temperatures were studied and then compared with those of bulk pMOSFET's.

At low temperatures, the suppression of short channel effects is greater in SOI devices than in bulk devices having the same channel doping. This occurs because, at low temperatures, L_{eff} in SOI devices increases more rapidly than in bulk devices and because the X_j of SOI devices is much smaller than that of bulk devices. The low-temperature carrier mobility of SOI devices increases more rapidly than that of

bulk devices. The above is supported by calculations which show that the total charge required to form an inversion layer is less for SOI devices than for bulk devices. The deviation from the theoretical value of the subthreshold swing of SOI devices at low temperatures may be due to the effect of the back gate interface, which is caused by changes in the Fermi level of Si with temperature.

The changes in the characteristics of SOI devices which are due to hot carrier effects are larger than in bulk devices. Also, the changes in the characteristics of SOI devices are larger at lower temperatures. Stressing in SOI devices traps electrons in the gate oxide and buried oxide. The trapped electrons in the buried oxide cause not only a positive shift in V_{th} but also increase g_m due to the charge coupling effect. From the viewpoint of reliability, the supply voltage of SOI devices must be reduced when they are operated at low temperatures. Provided the supply voltage can be reduced, ultra-thin-film SOI devices are promising candidates for deep-submicron MOSFET's operating at low temperatures.

References

1) Colinge, J. P.: Some properties of Thin-Film SOI MOSFET's *IEEE Circuits and Devices Magazine*, **3**, pp. 16-20 (1987).
2) Yoshimi, M., Hazama, H., Takahashi. S., Wada, T., Kato, K., and Tango, H.: Two- Dimensional Simulation and Measurement of High-Performance MOSFET's Made on a Very Thin SOI Film. *IEEE Trans. Electron Devices*, **ED-36**, pp. 493-503 (1989).
3) Kamgar, A., Hillenius, S. J., Cong, H. L, Field, R. L, Lindenberger, W. S., Celler, G. K., Trimble, L. E., and Sheng, T. T.: Ultra-Fast (0.5- μ m) CMOS Circuits in Fully Depleted SOI Films. *IEEE Trans. Electron Devices*, **ED-39**, pp. 640-647 (1992).
4) Laux, S. E.: Accuracy of an Effective Channel Length/External Resistance Extraction Algorithm for MOSFET's. *IEEE Trans. Electron Devices*, **ED-31**, pp. 1245-1251 (1984).
5) Kamgar, A.: Miniaturization of Si MOSFET's at 77K. *IEEE Trans. Electron Devices*, **ED-29**, pp. 1226-1228 (1982).
6) Yau, L. D.: A Simple Theory to Predict the threshold Voltage of Short-Channel IGFET's. *Solid-State Electronics*, **17**, pp. 1059-1063 (1974).
7) Watt, J. T., and Plummer, J. D.: Universal Mobility-Field Curves for Electrons and Holes in MOS Inversion Layers, *VLSI Tech. Dig.*, pp. 81-82 (1987).
8) Rideout, V. I. Gaensslen, F. H., and LeBlanc, A.: Device Design Consideration for Ion Implanted n-Channel MOSFET's. *IBM J. Res. Develop.*, **19**, pp. 50-59 (1975).
9) Ng, K. K., and Taylor, G. W.: Effects of Hot-Carrier Trapping in n- and p-Channel MOSFET's *IEEE Trans. Electron Devices*, **ED-30**, pp. 871-876 (1983).
10) Lim, H. K., and Fossum, J. G.: Threshold Voltage of Thin-Film Silicon-on-Insulator (SOI) MOSFET's. *IEEE Trans. Electron Devices*, **ED-31**, pp. 1244-1251 (1984).
11) Sukegawa, K., and Kawamura. S.: Effects of Hot Electron Trapping in Ultra-Thin-Film SOI/SIMOX pMOSFET's *IEICE Trans. Electron.*, **E75-C**, pp. 1484-1490 (1992).

Kazuo Sukegawa received the B.S., and M.S. degrees in electrical engineering from Hokkaido University, Sapporo, Japan, in 1985 and 1987, respectively. In 1987, he joined Fujitsu Ltd., Kawasaki, Japan, where he has been engaged in SOI device and process technology development.
He is a member of the Japan Society of Applied Physics.

Seiichiro Kawamura received the B.S. degree in Applied Physics from the University of Tokyo in 1974, and the M.S. degree in Solid State Physics from Princeton University, Princeton, NJ, USA, in 1978. He joined the IC Development Division, Fujitsu Ltd. in 1978, where he has been engaged in research and development of VLSI processing and device technologies. Since 1989, he has served on the technical program committees of the Symposium on VLSI Technology.

UDC 621.3.049.771.14:621.923

Influence of Silicon Surface Roughness on Time-Dependent Dielectric Breakdown

● Toshiro Nakanishi ● Sadahiro Kishii ● Akira Ohsawa

(Manuscript received December 7, 1992)

The effects of mechanochemical polishing on the surface flattening and the reliability of MOS diodes have been studied. Surface microroughness was observed with TOPO-2D, XTEM, STM, and AFM. The Fowler-Nordheim tunneling current, time-dependent dielectric breakdown (TDDB), surface state density, and flat band voltage under stress after fabricating MOS diodes on the polished wafers were measured.

It is confirmed that polishing leads to excellent TDDB characteristics, because the polishing reduces the surface roughness atomically, which decreases the tunneling current through the oxide. Polishing also lowers the surface state density and decreases a flat band voltage shift under constant current stress.

1. Introduction

State-of-the-art techniques in the study of surface morphology – e.g. high-resolution transmission electron microscope (HRTEM)[1)-4)], reflection electron microscopy (REM)[5),6)], spot profile analysis of low-energy electron diffraction (SPA-LEED)[7)-9)], scanning tunneling microscopy (STM)[10)], and atomic force microscopy (AFM)[10)] – have shed new light on what happens when tunneling occurs in the thin oxide layers in VLSI circuits.

Roughness at the Si-SiO_2 interface was observed with HRTEM[1),2)]. The peaks of the interface roughness for 4.1 nm oxides grown at 900 ℃ were about 1.4 nm. These degrade the dielectric breakdown strength. The interface roughness measured by SPA-LEED affects the fixed oxide charge density, interface state density, and Hall mobility[7)]. Ordinary SC-1 solution (NH_4OH : H_2O_2 : H_2O = 1 : 1 : 5) causes microroughness to degrade dielectric breakdown characteristics[10)].

Using STM, AFM, XTEM, and TOPO-2D (optical interferometry), we observed the surfaces of silicon wafers after different polishing treatments and investigated the effects of the roughness on the reliability of MOS diodes.

2. Wafer polishing

Figure 1 shows the mechanochemical polishing configuration. Mechanochemical polishing flattens silicon wafers without causing surface damage. Polishing uses a polyurethane pad and an alkaline solution containing a silica powder with particles 0.02 μm in diameter. The fine particles of colloidal silica mechanically remove

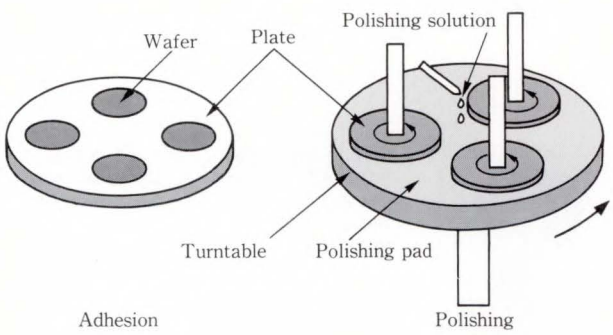


Fig. 1 – Wafer polishing system. The wafers are attached to the rotating plate and the polishing pads to the rotating turntable.

Table 1. The dependence of the surface roughness and the sillicon removal on the second polishing time

Second polishing time (min)	0	5	60
Silicon removal (nm)	0	31	483
Roughness (nm)	1.59	1.05	0.26

protruding atoms as the polishing plate rotates. The alkaline solution chemically etches the surface layer damaged by mechanical treatment. After polishing, the wafers' surfaces become mirror-like and flat on an atomic scale across the wafer surface.

In this experiment, we used 4-inch boron-doped 10-ohm-cm (100) CZ-Si wafers. The polishing solution was Glanzox 3 000 diluted 10 times, and the polishing pad was a Ciegal 7355-000. The pressure was 90 g/cm^2. After prepolishing on both sides, wafers were polished for 0, 5, and 60 minutes. Silicon removal is proportional to the polishing time (see Table 1).

3. Microroughness evaluation

3.1 Optical profilometer

The TOPO-2D optical profilometer can determine the roughness over 0.65 μm in a lateral direction using an interferometer. The incident light (λ = 650 nm) is split in two, and one beam is directed to the sample surface. The light intensity is changed by the surface roughness and the two beams are then recombined. The height of the protrusions is determined from the resulting light intensity. The detection limit was 0.12 nm and the repeatability 0.03 nm. The lateral resolution was 0.65 μm. To compare the roughness of each sample, the root mean square roughness (R_{rms}) was calculated by the following equation,

$$R_{rms} = \sqrt{\frac{1}{L}\int_0^L \{h(x)\}^2 dx}, \quad \cdots\cdots(1)$$

where L is the measured distance and $h(x)$ is the height at x (see Fig. 2).

The R_{rms} was 1.59 nm after the first polishing, but the surface became much flatter as the second polishing time increased (Table 1).

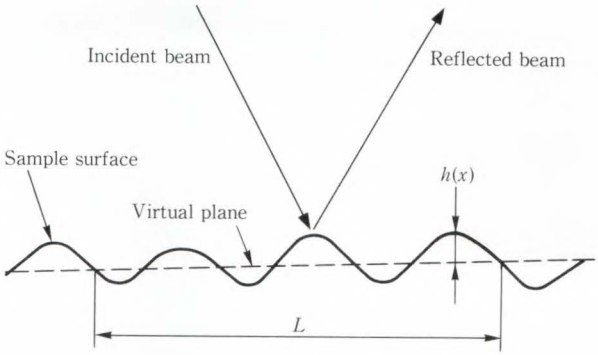


Fig. 2 – Measurement of the surface roughness with an optical interferometer.

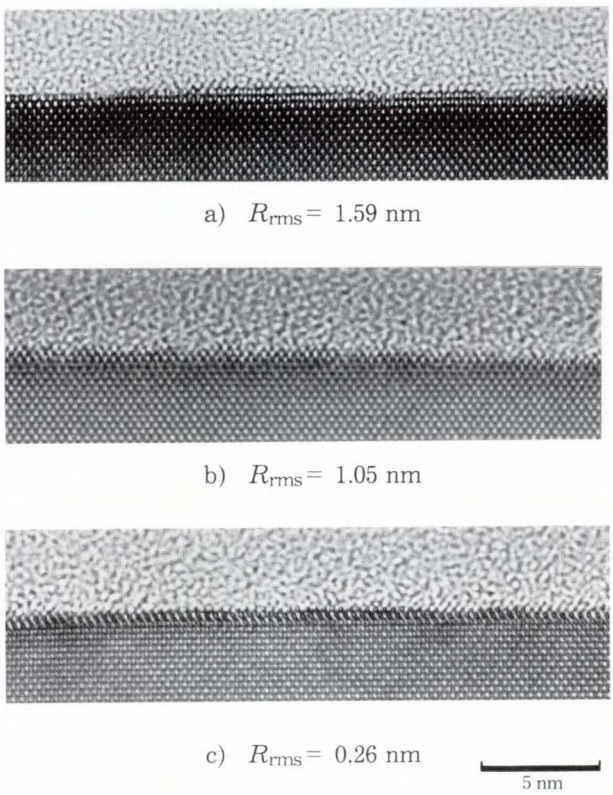
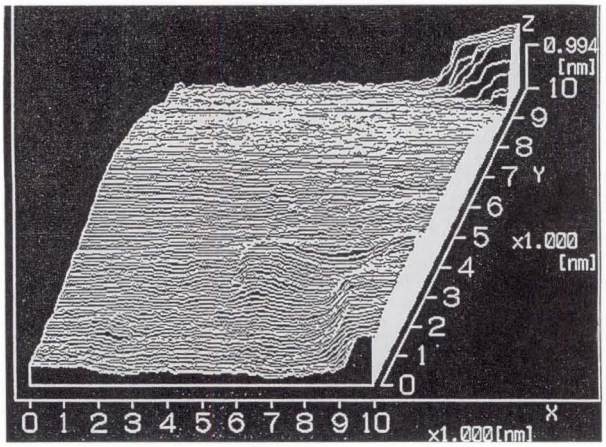


Fig. 3 – XTEM images of the interface roughness.

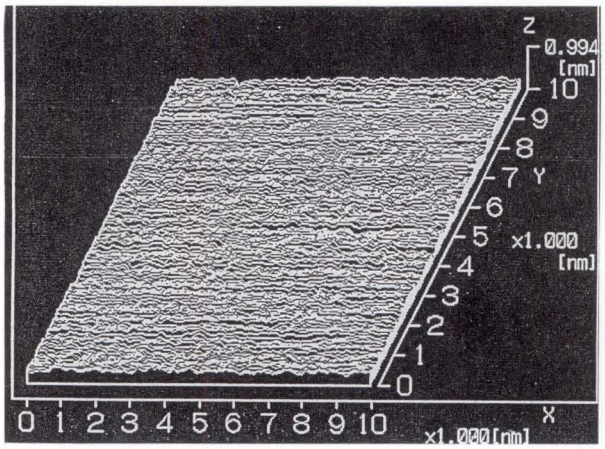
The mean distance between protrusions was 100 μm.

3.2 XTEM

High-resolution XTEM gives the position of each atom from its lattice image (see Fig. 3). The samples were thinned by argon ion milling for 30 to 35 hours. The acceleration voltage was 200 keV and the magnifying power 4 × 10^6. We observed undulations 2 to 3 atoms high at the



a) R_{rms} = 1.59 nm



b) R_{rms} = 1.05 nm

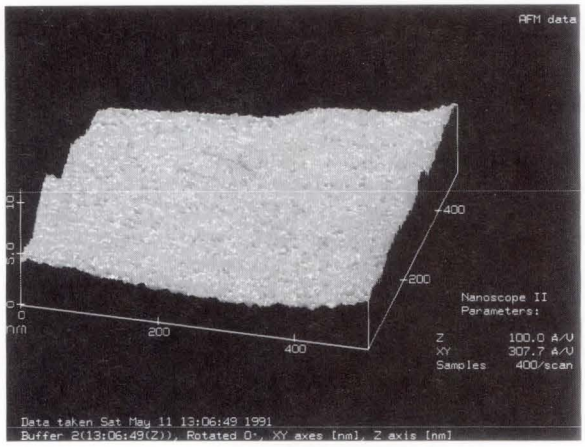
Fig. 4 – STM images of the silicon surface.
Scan area was 10 nm × 10 nm.

Si-SiO_2 interface of the 1.59-nm R_{rms} sample, but the interfaces of the 1.05- and 0.26-nm samples were comparatively flat.

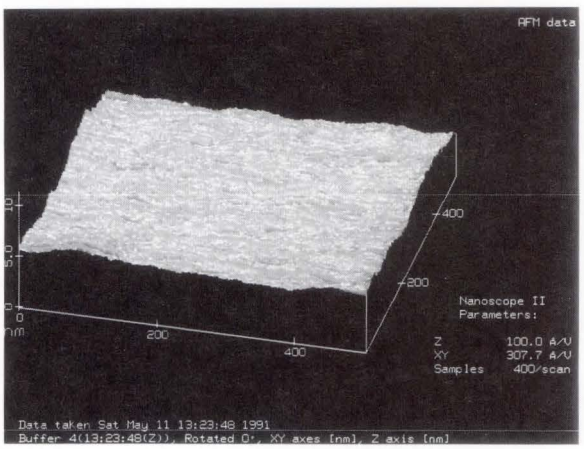
The XTEM samples were very thin, but they were still a few nm thick in the direction of the incident beam. The images are affected by the total thickness, so it is difficult to observe the true surface of the sample.

3.3 STM

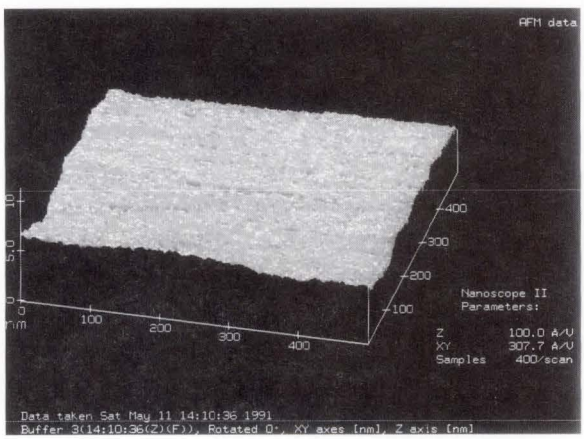
STM produces a two dimensional profile. The sample must be conductive, because STM monitors the tunneling current. The STM images were taken in air immediately after surface oxides were removed with a 5 percent HF solution (see Fig. 4). While the surface of the 1.05-nm sample was atomically flat and no protrusion was detectable, the surface of the 1.59-nm sample showed protruding atoms. The protrusions of over 1 nm high were seen. The



a) R_{rms} = 1.59 nm



b) R_{rms} = 1.05 nm



c) R_{rms} = 0.26 nm

Fig. 5 – AFM images of the silicon surface.
Scan area was 500 nm × 500 nm.

Table 2. Comparison of methods

Method	Interval (nm)	Height (nm)
TOPO-2D	$\sim 1 \times 10^5$	1.59 (R_{rms})
TEM	~ 25	~ 0.8
STM	~ 2	0.3 - 0.7

height of the protrusions and distance varied between them.

STM shows that mechanochemical polishing can, with time, flatten the wafer surface.

3.4 AFM

AFM uses the atomic force between the tip of the probe and the sample surface, therefore, the sample need not be conductive. The AFM observation was done immediately after oxide removal to avoid surface contamination. The scan area of AFM profiles (see Fig. 5) is 2.5×10^5 nm^2, which is 2 500 times as large as that of the STM profiles (see Fig. 4).

The maximum heights of the protrusions on the 1.59-, 1.05-, and 0.26-nm samples, which are shown in Figs. 5a), b), and c) respectively, were about 2-, 1-, and 1-nm. AFM also shows that polishing flattens the wafer surface of samples.

3.5 Comparison of methods

The estimated intervals between protrusions and heights of the 1.59-nm R_{rms} sample obtained using the above methods are listed in Table 2. Although the difference in protrusion heights is small, the distance between protrusions obtained by the various methods are quite different. To observe the roughness which affects the increase in electrical field in SiO_2, it is necessary to use the method which has the resolution less than the SiO_2 thickness.

TOPO-2D catches the roughness over an interval of approximately 100 μm. Even high-resolution XTEM merely catches protrusions about 25 nm apart which is more than the thickness of the oxide layer. XTEM showed that the oxide apparently formed uniformly along such a long-term protrusion. In this experiment, only STM reveals the effect of the roughness on the local thinning of the oxide layer on a nm order.

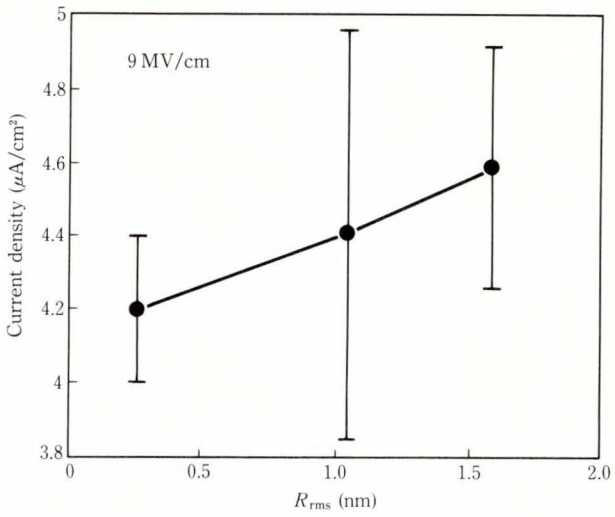


Fig. 6 – Fowler-Nordheim current for three surface roughnesses.

4. Electrical properties

To determine the effects of polishing on the reliability of the MOS diodes, diode arrays were fabricated on the wafers. After acid cleaning, oxide layers 11 nm thick were thermally grown at 900 ℃. Poly Si gates were deposited after gate oxidation. The samples were 5 percent H_2/N_2-annealed at 450 ℃, and gold contacts were evaporated onto the back of the wafers.

4.1 Fowler-Nordheim current

The Fowler-Nordheim (FN) current densities obtained with an electric field stress of 9 MV/cm were measured (see Fig. 6). The FN current through the oxides gradually decreased as the surface was flattened.

The FN current J_{flat} through a flat interface is described as[12]:

$$J_{flat} = AE^2 \exp(-\beta/E), \quad \cdots\cdots(2)$$

where A, E, and β are the proportional factor, the electrical field and the exponential factor respectively. Assuming that the surface roughness has a sinusoidal wave form, the FN current J_{rough} through a rough interface is expressed by the following equation[13],

$$J_{rough} = CJ_{flat}, \quad \cdots\cdots(3)$$

$$C \cong 1 + 1/12\,(\alpha^2 + 4\alpha + 6)\,\varepsilon^2, \quad \cdots\cdots(4)$$

where $\alpha = \beta/E$, $\varepsilon = \Delta/d_o$. Δ is the amplitude of the sinusoidal wave form and d_o is the average thickness of the oxide. The increasing factor of the surface roughness C can be estimated using Equation (4). Substituting the roughness height 2Δ shown in Table 2 into Equation (4) when $E = 9$ MV/cm, $\beta = 3.2 \times 10^8$ cm/MV, $d_o = 11$ nm, we calculated that $C = 1.16$ for $2\Delta = 0.8$ nm from XTEM image, $C = 1.12$ for $2\Delta = 0.7$ nm from the maximum value of the STM image, and $C = 1.02$ for $2\Delta = 0.3$ nm from the minimum value of the STM image.

The measured FN current shows that $J_{rough} = 1.09\,J_{flat}$ in the rough ($R_{rms} = 1.59$ nm) and the smooth ($R_{rms} = 0.26$ nm) wafers (see Fig. 6). An increasing factor of 1.09 reflects the morphology shown by the STM images. The oxide layer could not form evenly on the 1.59-nm sample observed in the STM image. The oxide might be thinned at the protrusions to increase the FN current. Mechanochemical polishing decreases the FN current by the atomically flattening effect.

4.2 Time-dependent dielectric breakdown (TDDB) characteristics

The TDDB characteristics for three samples with different surface roughnesses were measured under a constant field of 9 MV/cm (see Fig. 7). We found that the degradation starts earlier, and the cumulative failure is higher as the surface gets rougher. The time taken to reach a cumulative failure rate of 10 percent for the 0.26-nm sample was 1 500 times as long as that for the 1.59-nm sample. The relationship between the TDDB lifetime τ_{bd} and the applied field E is expressed by the following equation for the MOS diodes with uniform interfaces[14]:

$$\tau_{bd} = B \exp\{(\beta + H)/E\}, \quad \cdots\cdots(5)$$

where B and H are the proportional factor and the exponential factors in impact ionization coefficient respectively[15]. Although Equation (5) cannot express the difference between τ_{bd} for samples with different roughnesses quantitatively, the increase of E due to the local thinning reduces τ_{bd} significantly.

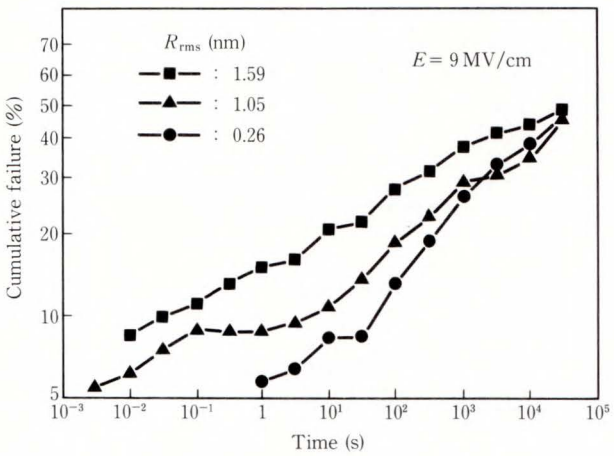


Fig. 7 – Time-dependent dielectric breakdown characteristics for three surface roughnesses.

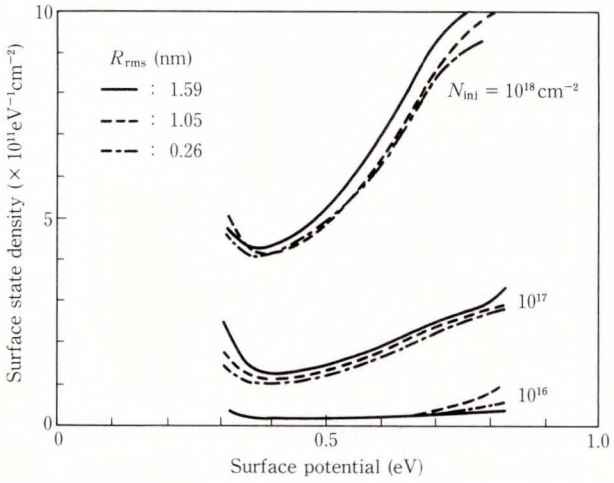


Fig. 8 – Surface state density for three surface roughnesses when constant current is injected.

We deduced the TDDB degradation mechanism from the roughness observations and the FN characteristics. Protrusions shown in the STM image (see Fig. 4) cause a local thinning of the oxide, which increases the FN current (see Fig. 6). When a high field is applied at this point, injected tunneling electrons are accelerated and become hot electrons. These hot electrons form hole traps in the oxide by impact ionization and, in time, oxide breakdown may develop at local thinning spots.

Mechanochemical polishing significantly improves the reliability of MOS diodes.

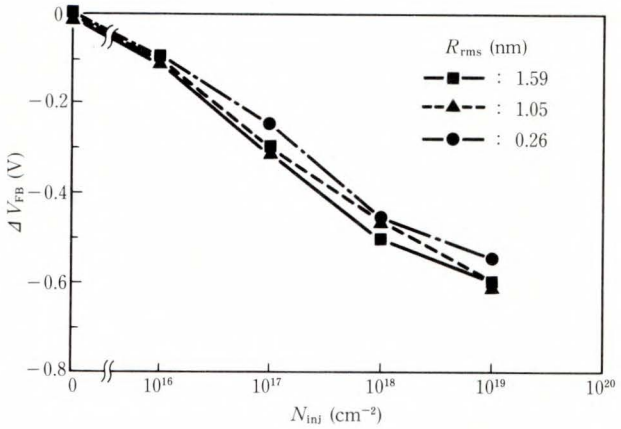


Fig. 9 – Flat band voltage shift under constant current stress.

4.3 Surface state density

The surface state densities of the surface roughnesses of the three samples were almost the same when the injected carrier density (N_{inj}) was 1×10^{16} cm^{-2} (see Fig. 8). Small differences appeared, however, when the N_{inj} was 1×10^{17} cm^{-2}; surface state density increases with surface roughness. When N_{inj} was 1×10^{18} cm^{-2}, the maximum surface state density at mid-gap was obtained by the roughest wafer. We think this increase on a rough wafer is caused by an aberration from the (100) crystal orientation at the rough interface. A (111) surface has a surface state density twice as high as a (100) surface under the same carrier injection[11]. Rough interfaces allow more dangling bonds to form at the Si-SiO_2 interface than smooth interfaces and this increases the surface state density.

4.4 Flat band voltage shift

Figure 9 shows the flat-band voltage shift (ΔV_{FB}) caused by constant current stress. Large roughness increases ΔV_{FB} for injected carrier densities over 10^{17} cm^{-2}. The accumulation of positive charges increases in the oxide layer with the large rough interface. Polishing is also effective on ΔV_{FB} decreasing.

5. Conclusion

We studied the effects of wafer polishing on the reliability of MOS diodes. We found that flattening the wafer surface by polishing decreases the Fowler-Nordheim current, and increases the TDDB life time under stress. If large protrusions are left on the surface, they form local thinning spots after oxidation. When a high field is applied at this point, injected tunneling electrons are accelerated and become hot electrons. These hot electrons form hole traps in the oxides by impact ionization and, in time, oxide breakdown may develop at local thinning spots.

We also found that polishing lowers the surface state density and decrease a flat band voltage shift under constant current stress. The increase of the surface state density on a rough surface is caused by aberrations in the (100) crystal orientation.

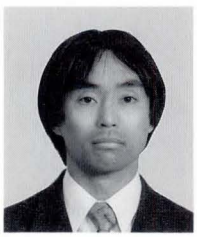
References

1) Carim, A. H. and Bhattacharyya, A.: Si/SiO_2 Interface, Roughness: Structural Observations and Electrical Consequences. *Appl. Phys. Lett.*, **46**, pp. 872-874 (1985).
2) Bhattacharyya, A., Vorst, C., and Carim, A. H.: Improved Electrical Breakdown Characteristics of Thin SiO_2 Films Processed by a Two-step Oxidation Treatment. 3rd VLSI Science and Technology, pp. 374-383 (1985).
3) Goodnick, S. M., Gann, R. G., Sites, J. R., Ferry, D. K., Wilmsen, C. W., Fathy, D., and Krivanek, O. L.: Surface Roughness Scattering at the Si-SiO_2 Interface. *J. Vacuum Sci. Technol.* **B1**, pp. 803-808 (1983).
4) Goodnick, S. M., Kerry, D. K., Wilmsen, C. W., Liliental, Z., Fathy, D., and Krivanek, O. L.: Surface Roughness at the Si (100)-SiO_2 Interface. *Phys. Rev.*, **B32**, pp. 8171-8186 (1985).
5) Honda, K., Ohsawa, A., and Toyokura, N.: Silicon Surface Roughness-Structural Observation by Reflection Electron Microscopy. *Appl. Phys. Lett.*, **48**, pp. 779-781 (1986).
6) Nakanishi, T., Honda, K., Kishii, S., and Ohsawa, A.: Degradation of Time-dependent Dielectric Breakdown Characteristics of MOS Capacitors by Silicon Surface Roughness. Proc. 1989 Int. Symp. VLSI Technol. Sys. Appl., pp. 79-84.
7) Hahn, P. O. and Henzler, M.: The Si-SiO_2 Interface: Correlation of Atomic Structure and

Electrical Properties. *J. Vac. Sci. Technol.*, **A2**, pp. 574-583 (1984).

8) Hahn, P. O., Grundner, M., Schnegg, A., and Jacob, H.: Correlation of Surface Morphology and Chemical State of Si Surfaces to Electrical Properties. *Appl. Surf. Sci.*, **39**, pp. 436-451 (1989).
9) Pietsch, G. J., Henzler, M., and Hahn, P. O.: Continuous Roughness Characterization from Atomic to Micron Distances: Angle-resolved Electron and Photon Scattering. *Appl. Surf. Sci.*, **39**, pp. 457-472 (1989).
10) Ohmi, T., Miyashita, M., Itano, M., Imaoka, T., and Kawanabe, I.: Dependence of Thin-oxide Films Quality on Surface Microroughness. *IEEE Trans. Electron Devices*, **ED-39**, pp. 537-545 (1992).
11) Miura, Y., Yamabe, K., Komiya, Y., and Tarui, Y.: The Effect of Hot Electron Injection on Interface Charge Density at the Silicon to Silicon Dioxide Interface. *J. Electrochem., Soc.* **127**, pp. 191-194 (1980).
12) Lenzlinger, M. and Snow, E. H.: Fowler-Nordheim Tunneling into Thermally Grown SiO_2. *J. Appl. Phys.*, **40**, pp. 278-283 (1969).
13) Honda, K. and Nakanishi, T.: Influence of Ni impurities at the Si-SiO_2 Interface on the MOS Characteristics. *Appl. Phys. Lett.* (to be published).
14) Chen, I. C. and Hu, C.: Accelerated Testing of Time-dependent Breakdown of SiO_2. *IEEE Electr. Dev. Lett.*, **EDL-8**, pp. 140-142 (1987).
15) Chen, I. C., Holland, S. E., and Hu, C.: Electrical Breakdown in Thin Gate and Tunneling Oxides. *IEEE Trans. Electron Devices*, **ED-32**, pp. 413-422 (1985).

Toshiro Nakanishi received the B.E. and M.E. degrees in electrical engineering from Kobe University, Kobe, Japan, in 1981 and 1983.
He joined Fujitsu Laboratories Ltd., Kawasaki, in 1983 and has been engaged in research of silicon dioxide and silicon material.
He is a member of the Institute of Electronics, Information and communication Engineers of Japan, and the Japan Society of Applied Physics.



Sadahiro Kishii received the B.E. degree in nuclear energy engineering from Osaka University, Osaka, Japan, in 1984.
He joined Fujitsu Laboratories Ltd., Kawasaki, in 1984 and has been engaged in research and development of semi-conductor polishing.
He is a member of the Japan Society for Precision Engineering and the Japan Society of Applied Physics.

Akira Ohsawa received the B. S. degree in material science from the University of Electro-Communications, Tokyo, Japan, in 1970. He received M.A. and Dr. degrees in Physics from Tohoku University, Sendai, Japan, in 1975. He joined Fujitsu Laboratories Ltd., Kawasaki, in 1976. and engaged in research and development of silicon crystal technology.
He is a member of the Japan Society of Applied Physics, and the Electrochmical Society.

UDC 621.376.24:621.396.946

QPSK Burst Demodulator for Satellite Communications Systems

● Makoto Uchishima ● Yoshiharu Tozawa ● Toshio Kawasaki

(Manuscript received November 30, 1992)

This paper describes a digital signal processing quadrature phase shift keying (QPSK) burst demodulator for satellite communications that uses a new technique which sequentially changes the receive filter. By changing the filter bandwidth according to the signal preamble pattern, the proposed technique improves the recovered carrier *S/N* and the recovered symbol clock *S/N*. The demodulator's low unique word miss probability (P_{miss}) and its low cycle-skip rate for the recovered carrier and symbol clocks at very low E_b/N_0 (3 dB or less) will ease the design of very small aperture terminal (VSAT) systems. A governmental communications application for the system is briefly discussed.

1. Introduction

Communication satellites such as the Japan Communication Satellites (JCSATs) and the Space Communication Satellites (SCSs)[1] have been launched in Japan recently. Their availability stimulated the development of very small aperture terminal (VSAT) communication systems.

These systems usually consist of many VSATs and a central hub station[2)-4)]. Although the VSAT's relatively small antenna of 0.8 to 1.2 meters, is a great cost advantage, it also makes the signal quality very low ($E_b/N_0 = C/2N = 3$ dB or less). Therefore, high coding gain forward error correction (FEC) is indispensable to improve the bit error rate (BER).

A high coding gain forward error corrector, such as a Viterbi decoder having a constraint length (k) of 7[5), 6)], requires a burst demodulator that operates at very low bit-energy to noise ratios (E_b/N_0), that has a low unique word miss probability (P_{miss}), and has a low cycle-skip rate (P_{cs})[7)-9)]. Since both carrier and clock synchronizing performance must be improved to meet these requirements, practical designs conventionally adopt the following three techniques:

1) loop-bandwidth-variable carrier recovery (CR) and symbol timing recovery (STR) circuits[10)-12)],
2) binary phase-shift keying (BPSK) demodulation instead of QPSK during the preamble to increase the E_b/N_0[11), 12)], and
3) a kick-off circuit for STR to prevent hang-up[12)].

Our digital signal processing (DSP) QPSK demodulator also uses these techniques. However, to take advantage of recent FEC advances such as a $k = 9$ Viterbi decoder[13)], a concatenated Reed-Solomon/Viterbi ($k = 7$) decoder[14), 15)], or a sequential decoder[16), 17)], additional techniques will be required to improve P_{miss} and P_{cs}.

To meet these strict requirements, we propose a technique to change the receive filter bandwidth according to the preamble pattern[18), 19)]. Theoretically, this technique should result in a recovered carrier S/N improvement of 1 dB and a recovered symbol clock S/N improvement of at least 3 dB. Measurement verified that the DSP demodulator improves P_{miss} by a factor 100 over that of a conventional DSP demodulator.

2. Major requirements

Of the major performance requirements fora

Table 1. Performance requirements

Modulation system	QPSK
Symbol rate	480k symbols/s
Carrier frequency error	± 4 kHz
Carrier cycle-skip rate	10^{-9} times/symbol at 0 dB E_b/N_0
Clock cycle-skip rate	10^{-9} times/symbol at 0 dB E_b/N_0
Unique word miss probability	10^{-7} times/symbol at 3 dB E_b/N_0

burst demodulator (see Table 1), we must first consider the cycle-skip rate. This should be on the order of 10^{-9} at an E_b/N_0 of 0 dB[3), 20)]. Thus, if the VSAT system uses a Viterbi decoder (k = 9, R = 1/2), the BER augmentation due to cycle skip should be less than 0.1 % at 0 dB. We must next consider P_{miss} and assume that its target value is 10^{-7} at an E_b/N_0 of 3 dB for a unique word length of 32 symbols and a tolerance of 14 bits. This means only one lost burst (2 kbits) per 86 hours at a 64 kb/s data rate. This P_{miss} value eases VSAT system design.

3. Conventional burst demodulation methods

This chapter briefly describes conventional carrier and clock synchronization methods. The burst data format is shown in Fig. 1 for reference.

1) CR and STR noise-bandwidth variation

During the data and unique word portion of the data burst, the CR noise-bandwidth upper bound (Bu_{cr}) is determined by the carrier cycle-skip rate. However, since a wide CR noise bandwidth (Bl_{cr}) is needed for initial carrier acquisition, we vary Bl_{cr} several times during the CR portion and the last Bl_{cr} must be below Bu_{cr} before the data portion begins. This is easily done using ROM-stored parameters to determine Bl_{cr}, and is easily implemented in the CR and STR blocks.

2) BPSK demodulation

The preamble consists of CR and STR portions. In the CR portion, the I and Q-channel signals are all 1s. During the STR portion, both channels consist of alternate 1s and 0s. Therefore, this is BPSK modulation, rather than QPSK. We use BPSK demodulation to increase E_b/N_0. Before the unique word is received, demodulation method must be changed from BPSK to QPSK by selecting the QPSK CR phase detector.

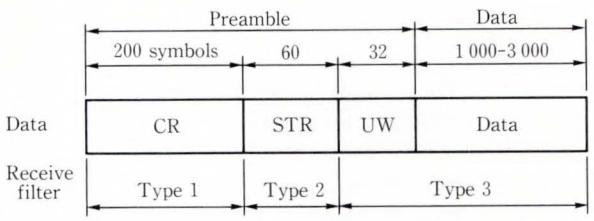


Fig. 1 – Burst data format.

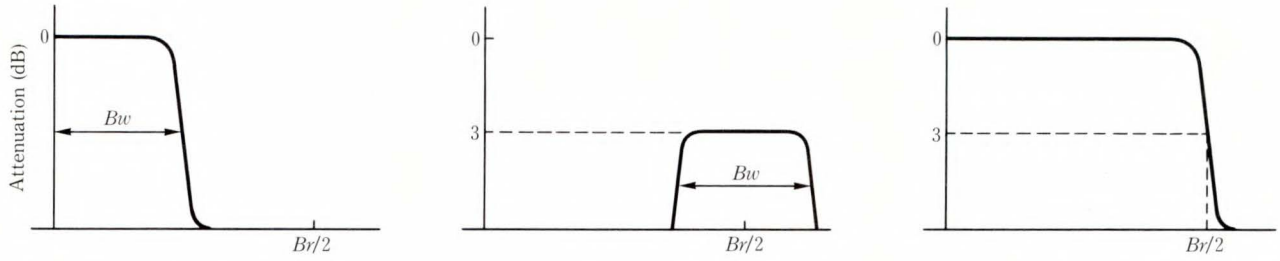
3) Kick-off

The STR block exhibits a hang-up phenomenon that causes STR hesitation before a phase lock is obtained and which is dependent on the initial phase condition between the receive and STR clocks. As a result, the decrease in P_{miss} is small even if the E_b/N_0 exceeds the critical level. Kick-off is often used to prevent hang-up, which can be detected by integrating the receive filter output signal at the STR timing over a 10-symbol period during the STR portion of the burst format. If hang-up is detected, the symbol clock is inverted to cancel it.

4. New burst demodulator

4.1 Receive filter changing technique

To increase the signal-to-noise ratios of the CR loop (SNl_{cr}) and STR loop (SNl_{str}), we propose a new technique to sequentially change the bandwidth of receive filter according to the preamble pattern. Three types of filters are used (see Fig. 2). Type 1 is a low-pass filter ($Bw << Br/2$) used during the CR portion. Type 2 is a bandpass filter ($Bw << Br/2$) used during the STR portion, and type 3 is a root-rolloff filter ($Bw = Br/2$) as is commonly used in a demodulator. Because the preamble consists of a stationary pattern, noise power is reduced without loss of signal power by using the receive filter appropriate for the signal preamble pattern.



a) Type 1 b) Type 2 c) Type 3

Fig. 2 – Receive filters.

Filter modification is easy to implement. Since a DSP demodulator usually uses a finite impulse response (FIR) receive filter, it is called a digital transversal filter (DTF). The DTF's characteristics are changed by changing its coefficients, and we have only to prepare three sets of coefficients, one for each receive filter.

1) Low-pass filter ($Bw << Br/2$) for the CR (type 1 filter)

Before proceeding, we must clarify the relationship between the signal-to-noise ratio of the receive filter output ($SNRi$) and the signal-to-noise ratio of the loop bandwidth (SNl). For carrier recovery, the signal-to-noise ratio of the CR loop (SNl_{cr}) is given by

$$SNl_{cr} = SNpd_{cr} \cdot \frac{Bw}{Bl_{cr}}, \qquad \cdots\cdots(1)$$

where Br is the symbol rate, Bw the receive filter bandwidth, and $SNpd_{cr}$ the signal-to-noise ratio of the CR phase detector output. This relationship is based on the work of F. M. Gardner[21]. $SNpd_{cr}$ depends on both $SNRi$ and on the order of the phase detector. During the CR portion, the CR has second-order nonlinearity because BPSK demodulation is used, and $SNpd_{cr}$ is given by

$$SNpd_{cr} = \frac{SNRi}{Rb(SNRi)} = \frac{SNRi}{4 \cdot (1 + \frac{1}{2 \cdot SNRi})}, \qquad \cdots(2)$$

where $Rb(SNRi)$ is the loss caused by nonlinearity[21].

We have already noted that phase detector nonlinearity degrades SNl_{cr} {see Equations (1) and (2)}. If there is no nonlinearity {$Rb(SNRi) = 1$}, the type 1 filter cannot improve SNl_{cr}, and SNl_{cr} is given by

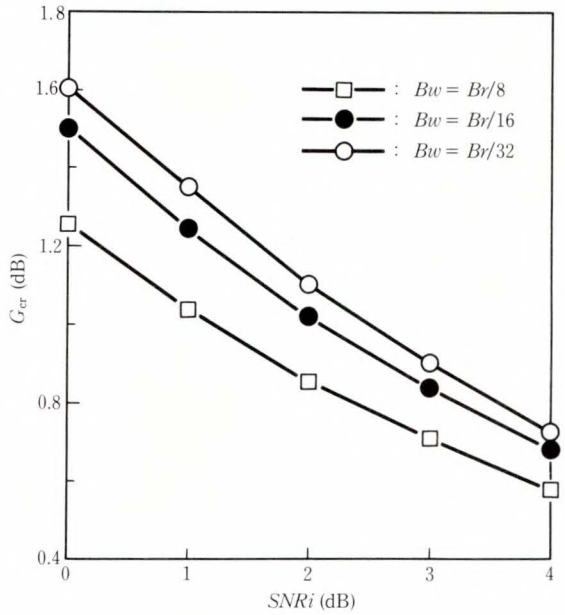


Fig. 3 – Rate of increase of SNl_{cr}.

$$SNl_{cr} = \frac{SNRi \cdot Bw}{Bl_{cr}} = \frac{SNRi' \cdot Br}{2 \cdot Bl_{cr}} = \frac{Ksn}{Bl_{cr}}, \qquad \cdots(3)$$

$$SNRi' = 2 \cdot E_b/N_0, \qquad \cdots\cdots(4)$$

where $SNRi'$ is $SNRi$ while using the type 3 filter ($Bw = Br/2$) with Ksn constant. Bw does not contribute to SNl_{cr}.

In practice, some second-order nonlinearity exists {see Equation (2)}, and SNl_{cr} is given by

$$SNl_{cr} = \frac{Ksn/Bl_{cr}}{Rb(SNRi)} = \frac{Ksn/Bl_{cr}}{Rb(Ksn/Bw)}. \qquad \cdots\cdots(5)$$

Bw contributes to SNl_{cr}. Therefore the rate of increase of SNl_{cr}, G_{cr}, is given by

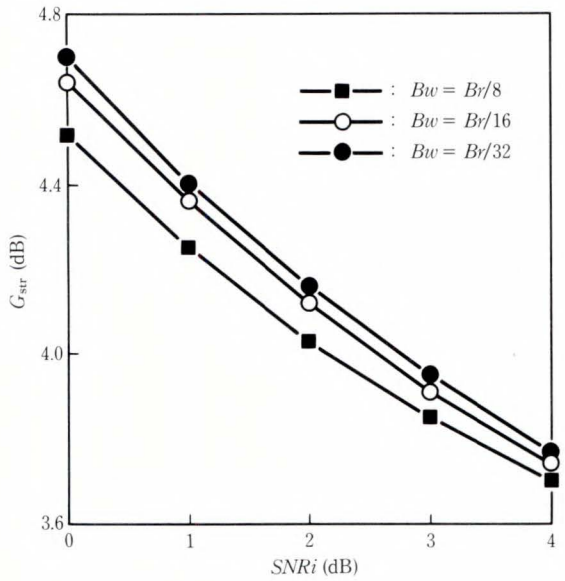


Fig. 4 – Rate of increase of SNl_{str}.

$$G_{cr} = \frac{Rb(SNRi')}{Rb(SNRi)} = \frac{\{1+(2\cdot SNRi')^{-1}\}}{\{1+(SNRi'\cdot Br/Bw)^{-1}\}}. \quad \cdots(6)$$

Figure 3 shows G_{cr}.

2) Passband filter ($Bw << Br/2$) for STR and kick-off circuits (type 2 filter)

STR has almost the same second-order nonlinearity as CR. Moreover, the attenuation of the type 3 filter at DC is 3 dB less than at $Br/2$ by virtue of the type 3 root-rolloff filter. Therefore, SNl_{str} is given by

$$SNl_{str} = 2\cdot SNpd_{str}\cdot Bw/Bl_{str}$$

$$= \frac{2\cdot Ksn/Bl_{str}}{Rb(SNRi)} = \frac{2\cdot Ksn/Bl_{str}}{Rb(2\cdot Ksn/Bw)}. \quad \cdots(7)$$

The rate of increase of SNl_{str}, G_{str}, is given by

$$G_{str} = \frac{2\cdot Rb(SNRi')}{Rb(SNRi)} = \frac{2\cdot\{1+(2\cdot SNRi')^{-1}\}}{\{1+(SNRi'\cdot Br/Bw)^{-1}\}}. \quad \cdots\cdots(8)$$

SNl_{str} increases by more than 3 dB. Figure 4 shows G_{str}.

4.2 Hardware configuration

The DSP demodulator consists of DTF, STR, CR, and kick-off circuits (see Fig. 5). The DTF, CR and STR circuits are implemented in highly integrated chips.

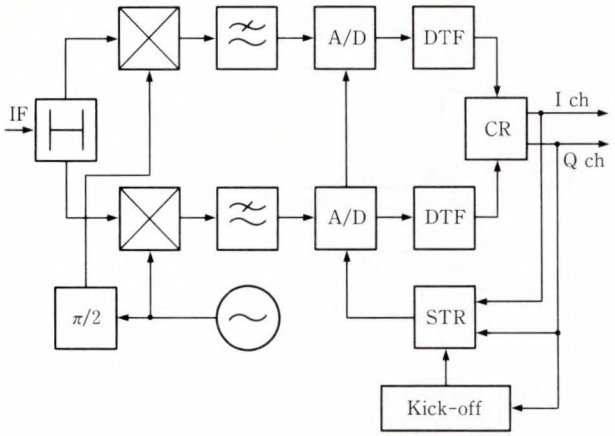


Fig. 5 – DSP demodulator.

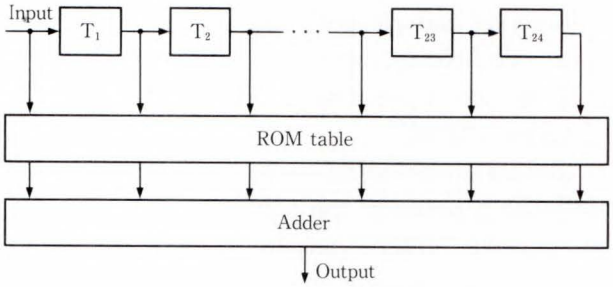


Fig. 6 – Digital transversal filter.

1) Digital transversal filter (DTF)

The DTF is a FIR filter consisting of an IC and two ROMs (see Fig. 6). The input signal rate is 4 samples/symbol and the output signal rate is 2 samples/symbol. The coefficients are symmetrical, and 12 out of 25 have the same value because the phase/frequency characteristic is linear. The ROMs hold three addressable sets of filter coefficients.

2) Symbol timing recovery (STR)

The frequency stability of the clock source oscillator must be very high for satellite communications. Since the clock frequency error is negligible, the STR need only recover the clock phase. The STR is a digital PLL controlling the dividing ratio N of a digital VCO (see Fig. 7). The STR consists of an IC and an external clock running at 64 times the symbol rate. The loop filter generates a control signal for the VCO when the accumulated clock phase error exceeds the $\pm K_{TH}$ threshold. The threshold is varied during the STR portion and, for

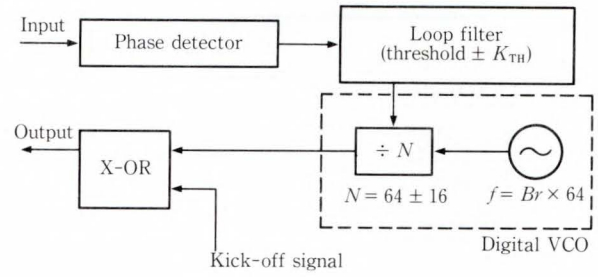


Fig. 7 – Symbol timing recovery.

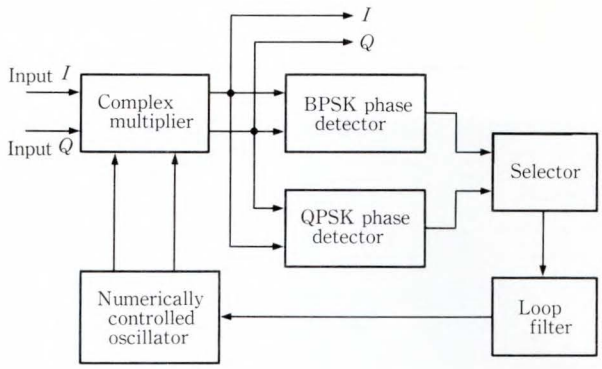


Fig. 8 – Carrier recovery.

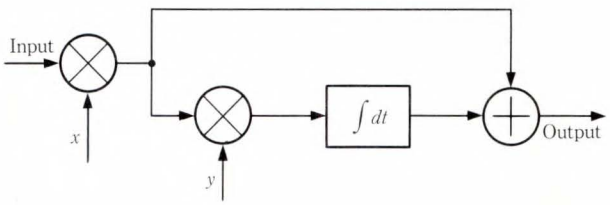


Fig. 9 – CR loop filter.

initial acquisition, the VCO output is exclusive-ORed with the kick-off signal and inverted.

3) Carrier recovery (CR)

The carrier recovery IC uses Costas PLL (see Fig. 8). Its loop filter is shown in Fig. 9. The noise bandwidth is a function of x and y, both of which are varied during the CR portion of the burst. The phase detector is switchable between BPSK and QPSK: BPSK is used during the preamble portion, QPSK during the data portion.

4.3 Performance

Figure 10 shows the P_{miss} performance of the demodulator at Bw_{cr} (the type 1 filter bandwidth) = $Br/12$, Bw_{str} (the type 2 filter bandwidth) = $Br/10$, Br = 480k symbols/s, Bu_{cr} (the CR upper bound of the noise bandwidth) = 1 kHz, and Bu_{str} (the STR upper bound of the noise bandwidth) = 17 kHz. Both Bu_{cr} and Bu_{str} are given by Equation (9)[21]. These Bu values mean that P_{cs} should be less than 10^{-9} at an E_b/N_0 of 0 dB[20].

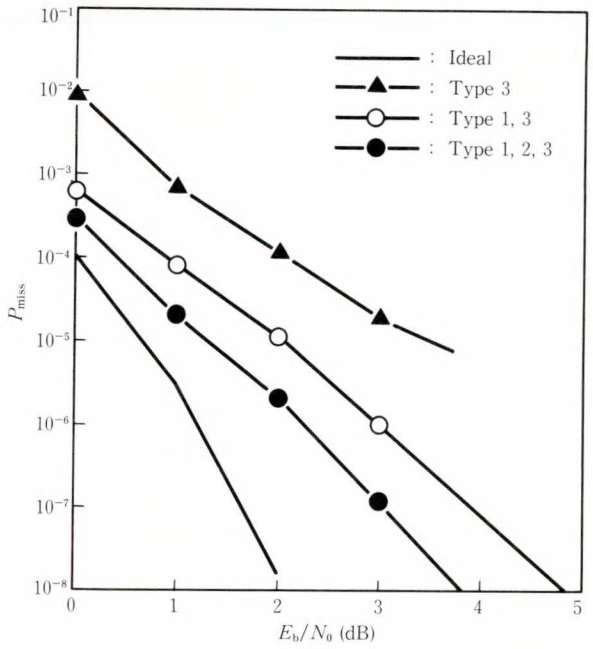


Fig. 10 – Improvements in unique word miss probability.

$$P_{cs} = \exp(-\pi \cdot SNl) \cdot Bu/Br. \quad \cdots\cdots(9)$$

A conventional demodulator with a type 3 filter only has hang-up problems because P_{miss} often jumps in value. A DSP demodulator with type 1 and type 3 sequentially changing filters eliminates hang-up and decreases P_{miss} to about 5 % at an E_b/N_0 of 3 dB. A DSP demodulator sequentially changing all three filter types decreases P_{miss} to about 0.5 % (see Fig. 10). Measured data meets the requirements given in Chap. 2.

5. Application to the VSAT system for city and prefectural governments

A VSAT system for city and prefectural governments which uses our DSP demodulator is shown in Fig. 11. Its system parameters are listed in Table 2. Figure 12 shows the DSP

Table 2. System parameters

	Voice facsimile	Broadcast		Channel control	
		Inbound	Outbound	Inbound	Outbound
Satellite	DA-FDMA	PA-TDMA	PA-TDMA	RA-TDMA	PA-TDMA
Modulation	QPSK burst	QPSK burst	QPSK continuous	QPSK burst	QPSK continuous
Symbol rate	35k	35k	35k	35k	35k
Information transmission rate	32 kb/s	32 kb/s	32 kb/s	32 kb/s	32 kb/s
Error correction	Viterbi $k = 7$	Viterbi $k = 7$	Viterbi $k = 7$	Viterbi $k = 7$	Viterbi $k = 7$

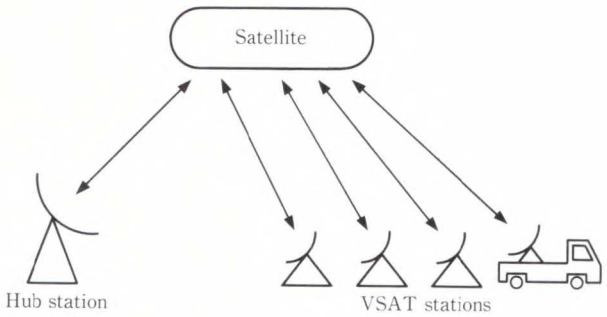


Fig. 11 – VSAT system.

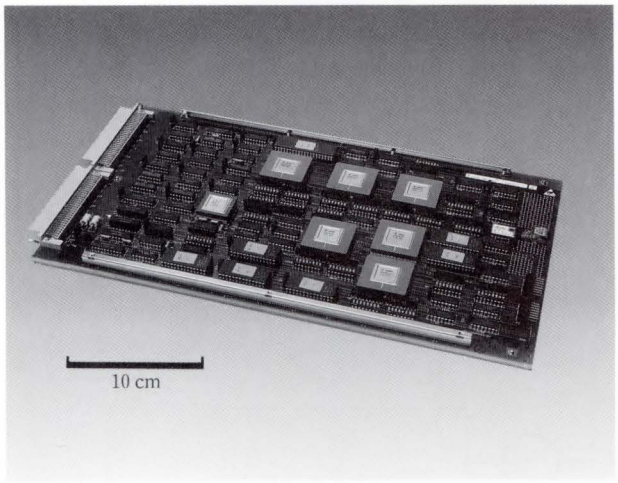


Fig. 12 – DSP demodulator.

demodulator.

The purpose of this VSAT system is to reliably transmit administrative information and gather local information at any time, even in a disaster.

One of the system's features is a voice activation technique[22] in the voice channel to enable efficient use of the satellite's power. Voice data is transferred as burst data, even if the satellite uses frequency division multiple access (FDMA). This system requires higher stability burst demodulators than is needed by other VSAT systems for low E_b/N_0 operation.

6. Conclusion

Achieving a low unique word miss probability (P_{miss}) at a low cycle-skip rate is a significant task in all satellite communications systems. We propose an effective solution to this problem in a technique which improves the initial acquisition of both the CR and the STR circuits. The method increases the S/N of the CR loop by about 1 dB and that of the STR by at least 3 dB. This means that the unique word miss probability of our demodulator is at least 100 times less than that of a conventional demodulator at an E_b/N_0 of 3 dB. The DSP demodulator enables VSAT systems to employ advanced FEC and voice activation techniques.

References

1) Simon, B.: Japan's Satellite Program. *Via Satellite*, **7**, 1, pp. 32-40 (1992).
2) Evans, B. G.: "Very small aperture terminals". Satellite communication systems, 1st ed., London, U. K., Peter Peregrinus Ltd., 1987, p. 223.
3) Tsujioka, T., Take, J., Valdez, C., and Shimamoto, S.: Experimental Projects on VSAT Communications. (in Japanese), *IEICE Tech.*

Rep., SAT91-93, pp. 13-18 (1991).

4) Darkin, J.: "Why Small Dishes ?". Satellite communications, 1st ed., Pinner, U.K., Online Conferences Ltd., 1984, pp. 63-74.
5) Yamashita, A., Nakamura, T., and Katoh, T.: A New Path Memory for Viterbi Decoders. *FUJITSU Sci. Tech. J.*, **25**, 1, pp. 37-43 (1989).
6) Kubota, S., Kato, S., Ishitani, T., and Nagatani, M.: Compact, High-Speed and High-Coding-Gain General Purpose FEC Encoder/Decoder－NUFEC CODEC－. Proc. ICC'89, Boston, U.S.A., 1989, pp. 798-803.
7) Wolejsza, Jr. I. J.: Simulation and Design of A Simple Burst Demodulator for A Random-Access TDMA Packet Terminal. Proc. ICDSC'86, Munich, Germany, 1986, pp. 683- 689.
8) Namiki, J., Ohtani, S., and Yasuda, Y.: 0dB Eb/No Burst Mode SCPC Modem with High Coding Gain FEC. Proc. ICC'86, Toronto, Canada, 1986, pp. 1792-1796.
9) Kato, S., Umehira, M., Miyo, T., and Seta, M.: Low C/N MODEM for Satellite TDMA Network Use. Proc. ICC'86, Toronto, Canada, 1986, pp. 1797-1802.
10) Suzuki, H., Takahashi, H., Tajima, M., and Kudoh, K.: MODEM and FEC LSIs for Highly Functional Compact Earth Station. Proc. GLOBECOM'87, Tokyo, Japan, 1987, pp. 281-285.
11) Nawata, H., and Otani, S.: Burst Demodulator for Offset QPSK. (in Japanese), Spring Natl. Conv. Rec. IEICE, 1991, p. SB-3-5.
12) Murakami, K., Miyake, M., Fuji, T., Moritani, Y., Fujino, T., and Takahata, F.: FEC Combined Burst-Demodulator for Business Satellite. Proc. GLOBECOM'87, Tokyo Japan, 1987, pp. 274-280.
13) Yamashita, A., Nakamura, T., Katoh, T., Moriwake, M., and Shimada, H.: Variable Constraint Length Viterbi Decoder LSI. (in Japanese), Spring Natl. Conv. Rec. IEICE, 1988, p. SB-3-7.
14) Fujino, T., Morita, Y., Miyake, M., and Shibuya, A.: Development of a Concatenated Reed-Solomon/Viterbi FEC Combined Modem and Its Field Test Via 14/11GHz Satellite. Proc. GLOBECOM'89, Dallas, U.S.A., 1989, pp. 1080-1087.
15) Kubota, S., Honda, S., Morikura, M., and Kato, S.: Concatenated Coding Scheme Employing Soft Decision for Outer Codes. Proc. ICC'91, Denver, U.S.A., 1991, pp. 221-225.
16) Peterson, W. W., and Weldon, JR. E. J.: "Sequential decoding". ERROR-CORRECTING CODES, 2nd ed., Cambridge, U.K., MIT PRESS, 1972, pp. 421-424.
17) Shimada, K., Yamashita, A Katoh, T., and Ageno, Y.: New Recovery Method for Sequential Decoder Buffer Overflow. Proc. GLOBECOM'86, Houston, U.S.A., 1986, pp. 1708-1712.
18) Uchishima, M., Tozawa, Y., Miyo, T., and Takenaka, S.: Burst Demodulator for Low E_b/N_0 Operation. Proc. ICC'91, Denver, U.S.A., 1991, pp. 226-230.
19) Lee, L-N., Sheoy, A., and Eng, M. K.: Digital Signal Processor-based Programmable BPSK/QPSK/offset-QPSK modems. *COMSAT Tech. Rev.*, **19**, 2, pp. 195-234 (1989).
20) Tozawa, Y., Furukawa, H., Takenaka, T., Miyo, T., and Takenaka, S.: Low Cycle Skipping Rate DSP Modem for Very Small Earth Stations. Proc. GLOBECOM'89, Dallas, U.S.A., 1989, pp. 1100-1104.
21) Gardner, F. M.: "Data Synchronizers". Phaselock Techniques, 2nd ed., New York, U.S.A., A Wiley-Interscience Pub., 1979, pp. 215-230.
22) Satoh, T.: "Voice Activation". Maritime Satellite Communication. (in Japanese), 1st ed., Tokyo, Japan, IEICE, 1986, p. 160.

Makoto Uchishima received the B.E. and M.E. degrees in aeronautics from Nihon University, Japan, in 1984 and 1986.
He joined Fujitsu Laboratories Ltd., Kawasaki, in 1986 and has been engaged in research and development of demodulators and FEC circuits for satellite communication systems.
He is a member of the Institute of Electronics, Information and Communication Engineers (IEICE) of Japan.

Yoshiharu Tozawa received the B.E. degree in electronics engineering from Nagoya Institute of Technology, in 1979, and the M.E. degree from Nagoya University, Nagoya, Japan in 1981.
He joined Fujitsu Laboratories Ltd., Kawasaki, in 1981 and has been engaged in research and development of microwave circuits and burst modem for satellite and mobile communications systems. He is a member of the Institute of Electronics, Information and Communication Engineers (IEICE) of Japan.

Toshio Kawasaki graduated from Kochi Technical College, Kochi, Japan in 1983.
He joined Fujitsu Ltd., Kawasaki, in 1983 and has been engaged in development of satellite communication systems.

UDC 621.382.333

Gas-Source MBE Growth of AlGaAs and GaAs for HBT Applications

● Toshio Fujii ● Hideyasu Ando ● Adarsh Sandhu ● Naoya Okamoto

(Manuscript received December 2, 1992)

Background impurities (carbon and oxygen) incorporation and the n-type doping of AlGaAs by gas-source molecular beam epitaxy (GSMBE) using triethylaluminum (TEAl) and trimethylamine alane as aluminum sources have been extensively studied; triethylgallium as the gallium source; cracked arsine as the arsenic source; and uncracked disilane as an n-type dopant source. A carbon-doped base ($p = 4 \times 10^{19}$ cm^{-3}, 92.5-nm thick, trimethylgallium as the carbon source) GaAs/AlGaAs heterojunction bipolar transistor was grown (TEAl as the aluminum source). The dc current gain of 45 was obtained at a current density of 4×10^4 A/cm^2 (4×5 μm^2 emitter). Device characteristics under current stress were found to be stable.

1. Introduction

Gas-source molecular beam epitaxy (GSMBE) using metalorganic group III and hydride group V sources has been used to grow III-V compound semiconductors. In particular, the ability to grow heavily carbon-doped GaAs (1×10^{21} cm^{-3}) using trimethylgallium ($(CH_3)_3Ga$, TMGa)[1] is one of the most attractive features of GSMBE for device applications. Carbon at high concentrations in GaAs has a low diffusion coefficient compared to other p-type dopants such as beryllium and zinc[2]. GSMBE is therefore promising in the growth of heavily carbon-doped heterojunction devices such as GaAs/AlGaAs heterojunction bipolar transistors (HBTs)[3, 4]. The ability to reproducibly control the n-type doping of AlGaAs has been a major concern in the practical application of GSMBE for the growth of HBTs. Major problems in the early stages of n-type doping of AlGaAs were:

1) the high background carbon concentration in AlGaAs grown using triethylaluminum ($(C_2H_5)_3Al$, TEAl), triethylgallium ($(C_2H_5)_3Ga$, TEGa), and arsine (AsH_3) as source gases[5] and
2) poor doping reproducibility using conventional hot solid dopant sources such as silicon and tin[6].

The background carbon concentration in AlGaAs grown by GSMBE using the source gases mentioned above was reduced to the order of 10^{17} cm^{-3} by optimizing growth parameters such as substrate temperature and the AsH_3 flow rate[5]. We reported that disilane (Si_2H_6) is a promising n-type cold gaseous dopant source for the GSMBE growth of n-AlGaAs[7, 8], and demonstrated the first growth of a GaAs/AlGaAs HBT by GSMBE using only gaseous sources[4]. Recently, the background carbon concentration was further reduced to the order of 10^{16} cm^{-3} by using new aluminum precursors such as trimethylamine alane ($(CH_3)_3NA1H_3$, TMAAl)[9, 10] and tri-isobutylaluminum ($(C_4H_9)_3Al$, TIBAl)[11, 12], which have led to excellent n-type doping controllability in practical application[10, 13].

In this paper, we describe our recent GSMBE study on the incorporation of background impurities of carbon and oxygen into AlGaAs which affects doping characteristics and the n-type doping of AlGaAs, using TEAl and TMAAl as aluminum sources, TEGa as the gallium source, cracked AsH_3 as the arsenic source, and uncracked Si_2H_6 as an n-type dopant source. We also report on the p-type doping of GaAs using TMGa as both the dopant

and the host material source. We finally report on the growth of carbon-doped base GaAs/AlGaAs HBTs using TEAl as the aluminum source, and discuss device characteristics and the stability against current stress[14].

2. Experiments

AlGaAs epilayers were grown on (100)-oriented semi-insulating GaAs substrates in a VG80H MBE growth chamber using a specially designed gas handling system[15]. All metalorganic sources were introduced into the growth chamber without the use of a carrier gas. The flow rates of the metalorganics were controlled by a differential pressure control method. Hundred percent AsH_3 was used as a group V source and was cracked at 1 100 ℃ in a low-pressure cracking cell. Si_2H_6, diluted to 10% in H_2, was used without precracking. The growth rate was 0.74 to 1.24 μm/h for $Al_xGa_{1-x}As$ (x = 0 to 0.40). The chamber pressure during growth was 5×10^{-5} to 1.5×10^{-4} torr. The carrier concentration of n-AlGaAs epilayers was determined by conventional C-V measurements. Atomic impurity concentrations were evaluated by secondary ion mass spectroscopy (SIMS) using Cs^+ primary ions. Calibration was done by comparison with silicon, oxygen, and carbon impurity-implanted MBE-grown GaAs and $Al_{0.3}Ga_{0.7}As$ as references.

3. Results and discussion

3.1 Dependence of impurity incorporation in AlGaAs on growth condition

We studied the incorporation of carbon and oxygen background impurities in AlGaAs by varing the grouth conditions. Figure 1 shows the variation of the background carbon concentration with substrate temperature, T_{sub}, and the AsH_3 flow rate for undoped $Al_{0.3}Ga_{0.7}As$ grown using TEAl (circles) and TMAAl (squares). The carbon concentration shows a weak V-shaped dependence on the substrate temperature when TEAl is used as the aluminum source at an AsH_3 flow rate of 2 sccm (closed circles). A minimum carbon concentration of 1.5×10^{18} cm^{-3} was obtained at 610 ℃. This substrate temperature dependence resembles that reported for

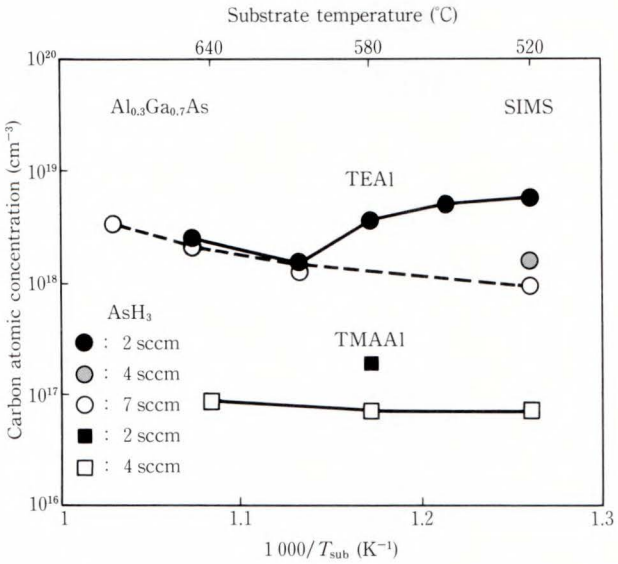


Fig. 1 – The variation of the background carbon concentration with T_{sub} and AsH_3 flow rate for undoped AlGaAs grown using TEAl and TMAAl.

GaAs[16], although AlGaAs shows a weaker variation. The carbon concentration decrease with increasing T_{sub} up to 610 ℃ is most likely due to the reduced number of incompletely decomposed metalorganic molecules {probably $Ga(C_2H_5)$ or $Al(C_2H_5)$}. Part of the third ethyl radical of TEGa or TEAl remains undecomposed on the substrate surface due to the low substrate temperature, leading to carbon incorporation. This model was used to explain carbon incorporation in GaAs epilayers when trimethylgallium ($(CH_3)_3Ga$, TMGa) is used as a gallium source[1), 17]. In the higher temperature range, the increased carbon concentration may be due to:

1) an increased rate of pyrolysis of the C-C bond in monoethylgallium or monoethylaluminum, which remain adsorbed on the substrate surface (β-methyl elimination reaction), and/or
2) the thermal decomposition of ethylradicals, which are by-products of TEAl or TEGa[18].

In the case of 1), the increased C-C bond dissociation {$M(C_2H_5) \longrightarrow M(CH_2) + CH_3$; M = Ga, Al} leaves molecules like $M(CH_2)$ on the substrate surface. The CH_2 radical part of $Ga(CH_2)$ or $Al(CH_2)$ adsorbed on the substrate surface then proceeds to occupy an arsenic site,

leading to carbon incorporation in the AlGaAs epilayer.

At a low T_{sub} of 520 ℃, carbon incorporation decreases notably with an increasing AsH_3 flow rate. A carbon concentration of 6×10^{18} cm^{-3} at 2 sccm was reduced to 1×10^{18} cm^{-3} at 7 sccm. Above 610 ℃, however, the carbon concentration was almost invariant for all AsH_3 flow rates, suggesting that the hydrogen from cracked AsH_3 (molecules and/or radicals) may react with the monoethylgallium or monoethylaluminum metal-carbon bond and enhance the dissociation of the third ethylradical bond, leaving metal atoms.

Carbon concentrations for TMAAl, are almost independent of substrate temperatures in the range studied. A minimum carbon concentration of 7×10^{16} cm^{-3} was obtained at T_{sub} between 520 and 580 ℃ and an AsH_3 flow rate of 4 sccm. The carbon concentration in AlGaAs using TMAAl decreased over an order of magnitude less than that using TEAl. The low carbon concentration over the wide substrate temperature range has the advantage of enabling us to choose the optimum temperature for practical device structures. Trimethylamine (TMA) appears easily removed from the Al atom and the C-N bond is thermodynamically more stable than the C-C bond. Decreasing the AsH_3 flow rate to 2 sccm increased the carbon concentration, suggesting that the carbon concentration could be effectively reduced by increasing the AsH_3 flow rate.

Figure 2 shows the dependence of the oxygen concentration on substrate temperature for undoped $Al_xGa_{1-x}As$ grown using TMAAl (open squares, x = 0.34, AsH_3 = 4 sccm) and TEAl (closed circle, x = 0.3, AsH_3 = 4 sccm), together with previously published data by GSMBE using the same combination of source gases (closed squares, TMAAl, TEGa and AsH_3, x = 0.42, AsH_3 = 5 sccm)[19]. Present data using TMAAl shows a constant oxygen concentration of 4×10^{17} cm^{-3}, the same as that using TEAl, and about an order of magnitude lower than previously reported data[19]. The oxygen in our experiments was supposed to come from the gas supply line, not from TEAl and TEGa as

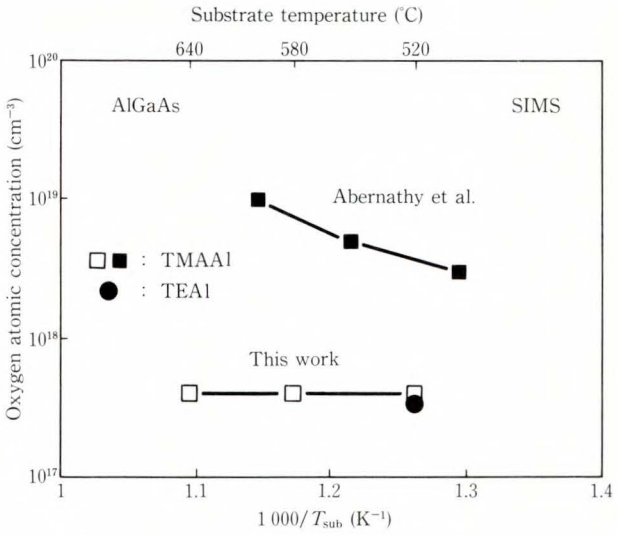


Fig. 2 – The dependence of the oxygen concentration on T_{sub} for undoped AlGaAs. Abernathy et al. reported for AlGaAs grown by using the same combination of source gases (TMAAl, TEGa and AsH_3)[19].

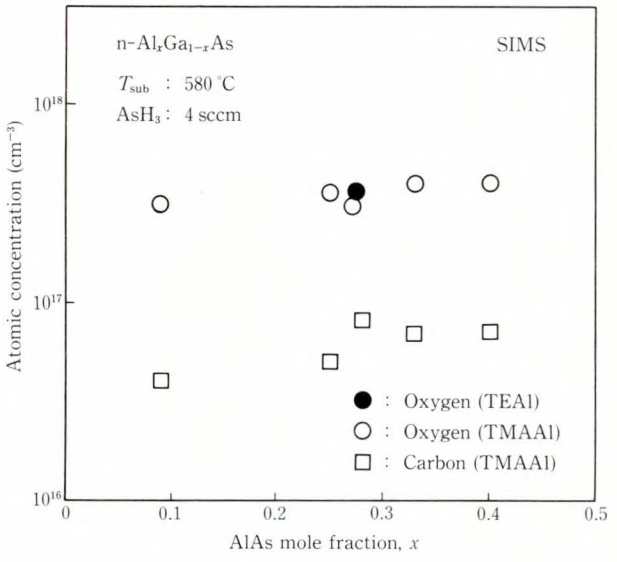


Fig. 3 – The variation of oxygen and carbon concentration in n-AlGaAs with the AlAs mole fraction.

described in the previous report[20].

Figure 3 shows the variation of carbon concentration (open squares) and oxygen concentration (open circles) in Si-doped n-AlGaAs using TMAAl with the AlAs mole fraction. Epilayers were grown at a substrate

temperature of 580 ℃ and an AsH_3 flow rate of 4 sccm. Data using TEAl as the Al source (closed circle) is also plotted for comparison. The oxygen concentration was between 3×10^{17} and 4×10^{17} cm^{-3} over the entire AlAs mole fraction range investigated, which is the same as that using TEAl.

The carbon concentration increases gradually with the increasing AlAs mole fraction. This is seen in GSMBE-grown AlGaAs using TEAl or TIBAl as an Al source gas[11), 16)], but the carbon concentration shown in Fig. 3 is over an order of magnitude less than that using TEAl (10^{18} cm^{-3}).

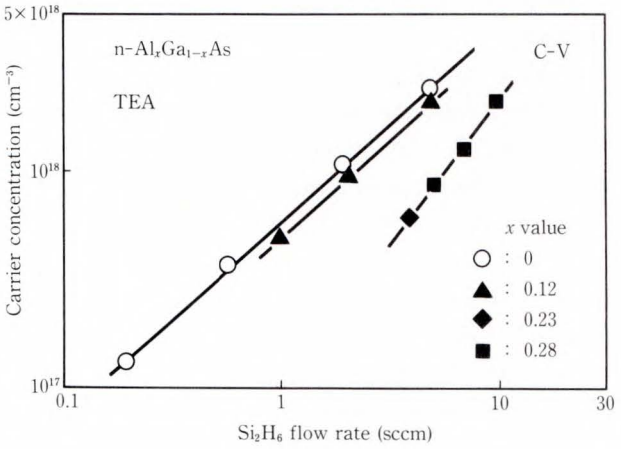
Oxygen and carbon concentrations in n-GaAs grown under the same conditions were below the SIMS detection limit, that is, less than 5×10^{16} cm^{-3} for oxygen and less than 2×10^{16} cm^{-3} for carbon.

3.2 Doping characteristics of n-AlGaAs using Si_2H_6

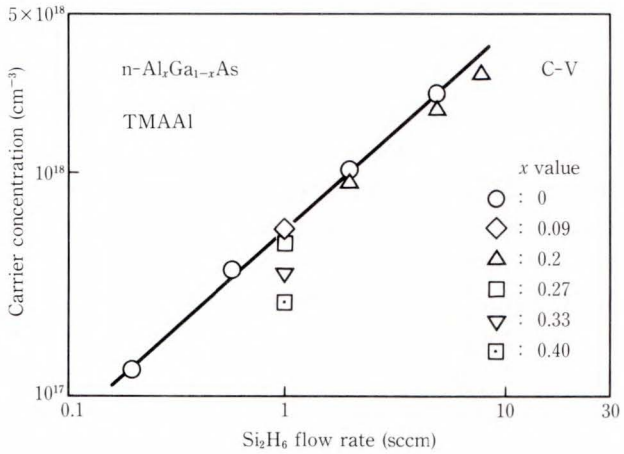
The dependence of carrier concentration of n-AlGaAs on Si_2H_6 flow rate, Si incorporation efficiency into AlGaAs, and the electrical activation efficiency of n-AlGaAs were studied. Figures 4a) and b) show the variation of the carrier concentration with diluted Si_2H_6 flow rate of n-$Al_xGa_{1-x}As$ (x = 0-0.40) grown using TEAl and TMAAl. The substrate temperature was 580 ℃ and the AsH_3 flow rate was 4 sccm. The carrier concentration plotted is normalized to a growth rate of 1 μm/h. Results showed that GaAs can be controllably doped from 1×10^{17} to 3×10^{18} cm^{-3} at normal growth temperatures. Using TEAl as shown in Fig. 4a), it can be seen that the carrier concentration of an $Al_xGa_{1-x}As$ (x = 0.23-0.28) epilayer is always much less than the corresponding GaAs epilayer for the same Si_2H_6 flow rate. This difference is attributable to the larger background carbon concentration in AlGaAs than in GaAs.

Our results also show that Si_2H_6 dissociates much more readily than SiH_4, of which pyrolysis was enhanced by precracking the SiH_4 for a tantalum filament to obtain electron concentrations above $10^{16}cm^{-3}$ [21)].

In contrast, with TMAAl of Fig. 4b), the



a) TEAl



b) TMAAl

Fig. 4 – The variation of the carrier concentration with Si_2H_6 flow rate of n-AlGaAs grown using TEAl and TMAAl.

carrier concentration of n-$Al_xGa_{1-x}As$ (x = 0.09-0.27) was reproducibly controlled between 5×10^{17} and 3×10^{18} cm^{-3} by varying the Si_2H_6 flow rate from 1 to 8 sccm. The carrier concentration of AlGaAs showed the same Si_2H_6 flow rate dependence as that of GaAs at the aluminum composition of 0-0.27. That is because the use of TMAAl significantly reduced the concentration of carbon acceptors, as shown in Figs. 1 and 3. The carrier concentration decreased when aluminum composition increased from 0.27 to 0.40 at a constant Si_2H_6 flow rate of 1 sccm. However, compensation by carbon acceptors is thought to affect the aluminum

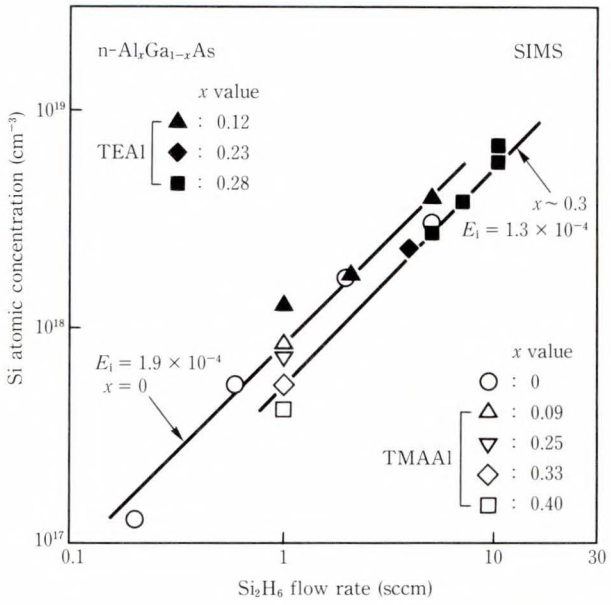


Fig. 5 – The variation in Si atomic concentration with the Si_2H_6 flow rate for n-AlGaAs using TEAl and TMAAl.

composition dependence of the carrier concentration negligibly, because the hole concentration originating from carbon acceptors is low (10^{16} cm^{-3}) even at $x = 0.4$. The carrier concentration decrease for x greater than 0.27 is discussed later.

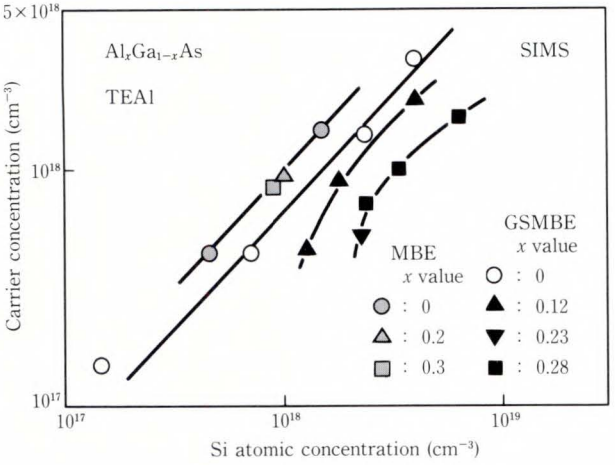
To investigate the depedence of Si atomic concentration of n-AlGaAs on the Si_2H_6 flow rate, SIMS measurements were performed for the same samples in Figs. 4a) and b). Figure 5 shows variations in Si atomic concentration with the Si_2H_6 flow rate for n-$Al_xGa_{1-x}As$ (x = 0-0.4) grown at a T_{sub} of 580 ℃ and an AsH_3 flow rate of 4 sccm using TEAl (closed symbols) and TMAAl (open symbols). The Si atomic concentration plotted is normalized to a growth rate of 1 μm/h. The Si atomic concentration in the GaAs epilayer (open circles) is almost proportional to the Si_2H_6 flow rate. The Si incorporation efficiency, E_i, was estimated to be 1.9×10^{-4}, using the relationship:

$$\frac{\text{Si atomic concentration}}{\text{Ga atomic concentration}} = \frac{E_i \times (Si_2H_6 \text{ flow rate} \times 10\% \times 2)}{\text{Calculated TEG flow rate}}.$$

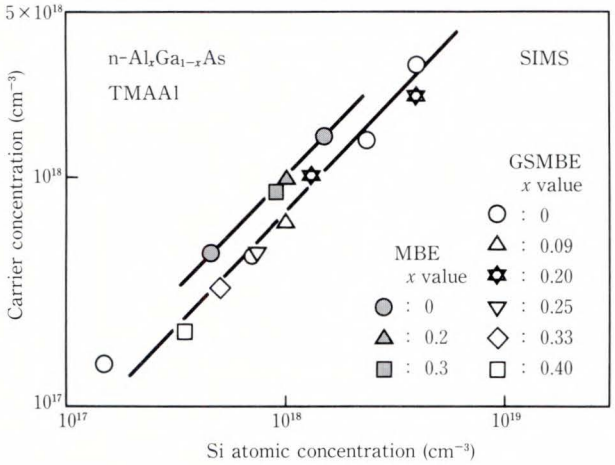
The corresponding E_i for AlGaAs epilayers with $x \sim 0.1$ is the same as that of GaAs, but E_i decreases from 1.9×10^{-4} to 1.3×10^{-4} when the Al content was increased from 0 to 0.3. The incorporation of Si into $Al_xGa_{1-x}As$ ($x > 0.2$) decreases with an increasing AlAs mole fraction. This was observed for TEAl and TMAAl. The decreased carrier concentration of n-$Al_xGa_{1-x}As$ (x = 0.27-0.40) in Fig. 4b) is probably due to decreased Si atomic concentration. However, this behavior of Si incorporation into AlGaAs differs from that of low-pressure (LP) MOVPE (78 torr) in which the Si atomic concentration does not depend on the Al content of epilayers[22]. This may be due differences in pressure during growth. That is, in LP-MOVPE, Si_2H_6 thermal decomposition proceeds before Si_2H_6 molecules reach the growth surface. In GSMBE (10^{-4} to 10^{-5} torr), Si incorporation is limited by the decomposition of Si_2H_6 only at the growth surface, and so decomposition is sensitive to the AlAs mole fraction of the growth surface layer.

The dependence of the carrier concentration of n-$Al_xGa_{1-x}As$ (x = 0-0.40) on the Si atomic concentration grown at a T_{sub} of 580 ℃ and an AsH_3 flow rate of 4 sccm using TEAl (closed symbols) and TMAAl (open symbols) was shown in Figs. 6a) and b) by using data of Figs. 4a), b) and 5. Data for n-GaAs and n-$Al_xGa_{1-x}As$ (x = 0.2-0.3) grown by conventional MBE are also plotted for comparison (shaded symbols). The carrier concentration for MBE-grown epilayers is clearly proportional to the Si atomic concentration, and all incorporated Si atoms are electrically active as donors; that is, the activation efficiency is one. The carrier concentration of GSMBE-grown GaAs from 1.5×10^{17} cm^{-3} to 3×10^{18} cm^{-3} (open circles) is almost proportional to the Si atomic concentration and more than 60 % of the Si atoms in GaAs are electrically activated. In Fig. 6a), the carrier concentration drop of AlGaAs below 10^{18} cm^{-3} is mainly due to carbon acceptor compensation. The reason for low activation efficiency in the carrier concentration region greater than 10^{18} cm^{-3} is not clear.

In Fig. 6b), note that the carrier concentration of AlGaAs grown using TMAAl shows

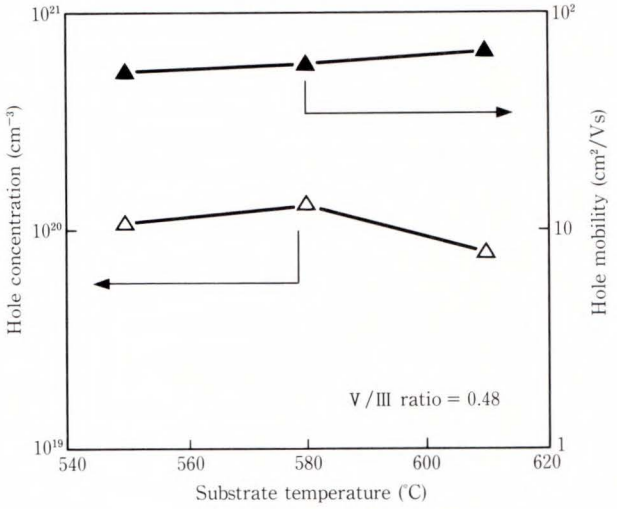


a) TEAl

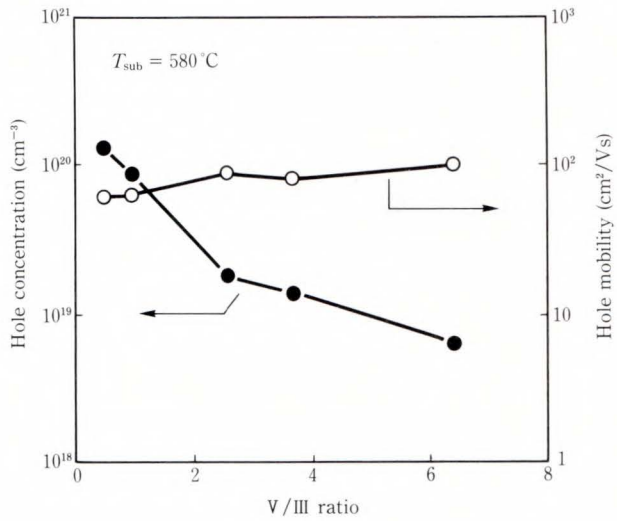


b) TMAAl

Fig. 6 – The dependence of the carrier concentration of n-AlGaAs (x = 0-0.4) on the Si atomic concentration using TEAl and TMAAl.



a) Substrate temperature



b) V/III ratio

Fig. 7 – The variation of GaAs hole concentration with substrate temperature and V/III ratio.

the same linear dependence as GaAs, quite different from that of AlGaAs using TEAl in which the carrier concentration is always smaller than that of GaAs at the same Si atomic concentration as shown in Fig. 6a). The activation efficiency of Si is more than 60 % for AlGaAs grown using TMAAl and most incorporated Si atoms seems to be activated as shallow donors even at concentrations up to 3×10^{18} cm^{-3}. These results indicate that the doping controllability of n-AlGaAs is significantly improved using TMAAl, particularly in the case of lower carrier concentrations and higher AlAs mole fractions.

3.3 p-Type doping of GaAs using TMGa

TMGa is a promising p-type dopant source for the GSMBE growth of GaAs[1]. We studied the p-type doping of GaAs using TMGa as both the dopant and the host material source and AsH_3 as the arsenic source. Figures 7a) and b) show the variation of the p-type GaAs epilayer hole concentration and mobility with substrate temperature and V/III ratio, respectively. All the p-type epilayer had excellent mirror surface

morphologies for all growth conditions studied. The epilayer mobilities varied from 100 cm^2/Vs at a hole concentration of 6.3×10^{18} cm^{-3} to 60.5 cm^2/Vs at a hole concentration of 1.3×10^{20} cm^{-3}. The GaAs hole concentration is seen to be relatively insensitive to changes in the substrate temperature between 550-610 ℃. However, the hole concentration is strongly dependent on the V/III rario as shown in Fig. 7b). The epilayer mobilities are considered to be comparable or better than p-type GaAs epilayers having similar doping levels grown by other growth methods. The strong V/III ratio dependence and relative independence with changes in growth temperature under these growth conditions, implies that the incorporation of the acceptor impurities (carbon) is limited by the reaction of the TMGa with AsH_3 by-products, and not only pyrolitically limited as was reported in the case of TMGa-As_4[23]. Our results show that the use of TMGa-AsH_3 the combination allows the easy and reproducible control of the epilayer hole concentration by simply adjusting the TMGa and AsH_3 flow rates, which is simpler and more accurate than the adjustment of substrate temperature. Thus, by the appropriate choice of growth conditions, the GaAs epilayer hole concentration can be easily varied between 6.3×10^{18} and 1.3×10^{20} cm^{-3}.

3.4 Growth and characterization of HBT structure

We grew a carbon-doped base GaAs/AlGaAs HBT using TEAl as an aluminum source. The HBT structure is shown in Fig. 8. The entire structure was grown at a substrate temperature of 580 ℃ with optimization of least-carbon compensation in the AlGaAs emitter layer. The $Al_{0.2}Ga_{0.8}As$ emitter layer doped with silicon from Si_2H_6 to a carrier concentration of 9×10^{17} cm^{-3} has an abrupt junction. A GaAs base layer was doped with carbon to a carrier concentration of 4×10^{19} cm^{-3} using TMGa as both the Ga source and the carbon source. The base layer is 92.5 nm thick. A 7.5-nm thick undoped GaAs spacer layer was grown between the emitter and base layers. The AsH_3 flow rate is kept constant at 4 sccm except for the base layer, for which the flow rate is reduced to 3 sccm to increase the carbon incorporation from TMGa. The carbon concentration is well controlled by varying an AsH_3 flow rate as previously reported[7]. Detailed growth conditions have been described in another paper[24]. Conventional wet chemical etching and lift-off were used to fabricate HBTs. A CVD-SiO_2 film was deposited for surface passivation. The emitter and collector ohmic contact metals were AuGe/Au, alloyed at 400 ℃ in N_2 gas. The base ohmic contact metal was nonalloyed Cr/Au.

Epitaxial layer	Carrier concentration (cm^{-3})	Thickness (nm)
n^+-GaAs	5×10^{18}	50
n-GaAs	2×10^{18}	230
N-$Al_{0.2}Ga_{0.8}As$	9×10^{17}	150
i-GaAs	—	7.5
p^+-GaAs	4×10^{19}	92.5
n-GaAs	1×10^{17}	400
n^+-GaAs	3×10^{18}	500
SI GaAs substrate		

Fig. 8 – Carbon doped base HBT structure.

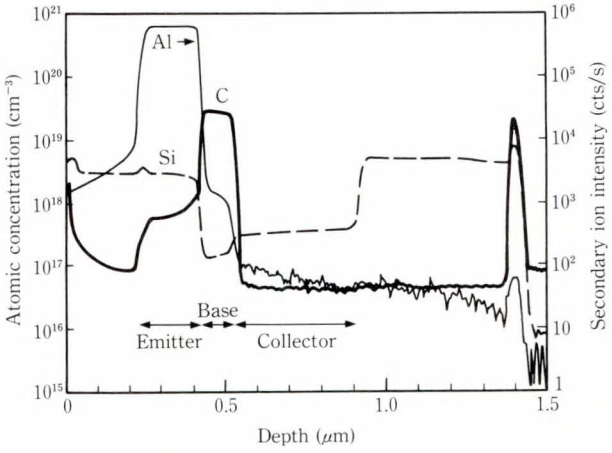


Fig. 9 – SIMS depth profile for a HBT structure.

Figure 9 shows a SIMS depth-profile of the HBT structure. Carbon, silicon, and matrix elements were measured. The atomic carbon concentration in the base layer is almost the

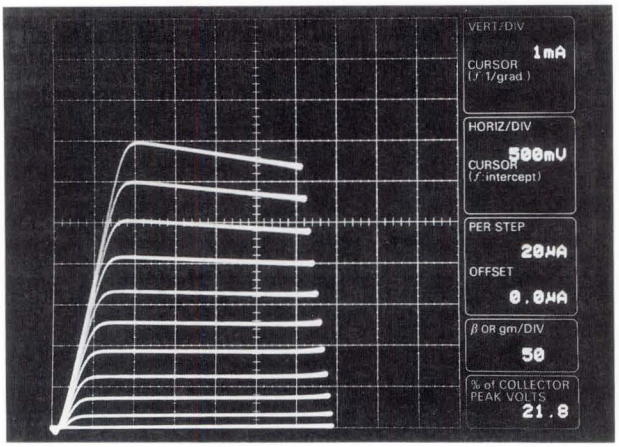


Fig. 10 – Common-emitter I - V characteristics of the HBT. The vartical scale is 1 mA/division and the horizontal scale is 500 mV/division. The base current is 20 μA/step.

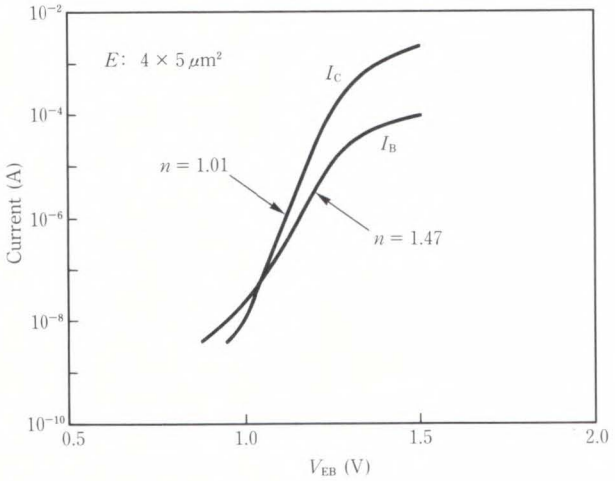


Fig. 11 – Gummel plots for a carbon doped base HBT.

same as the carrier concentration of 4×10^{19} cm^{-3}. The carbon profile shows a sharp drop at the emitter-base interface within the depth resolution of the SIMS, in this case, about 7.5 nm. This suggests that a well-defined interface was obtained using Si from disilane as an emitter dopant source and carbon from TMGa as a base dopant source.

Figure 10 shows the common-emitter I-V characteristics of the HBT having an emitter of $4 \times 5\,\mu m^2$ A dc current gain of 45 was obtained at a collector current density of 4×10^4 A/cm^2 and an emitter-collector voltage of 2.5 V, suitable for practical IC fabrication. The transistor had a turn-on voltage of about 0.2 V.

Figure 11 shows the Gummel plot of the transistor. The base current ideality factor was determined to be 1.47, suggesting that the generation-recombination current at the emitter-base junction is very small, and the emitter-base interface is well-defined. The ideality factor for the emitter-base diode was also measured to be 1.12, indicating high-quality GSMBE growth at the emitter-base junction. The superior quality of our HBT is seen when comparing the ideality factors obtained with Tin of 1.4[3] or TESn of 1.31[25] as emitter dopant sources. This advan-tage may be ascribed to the use of Si from Si_2H_6 as an emitter dopant source, which is a well-behaved impurity in AlGaAs.

We studied the electrical stability of a carbon-doped-base HBT under current stress[14]. The Be-doped-base HBT grown by conventional MBE reportedly shows a large device degradation due to Be diffusion from the base to the emitter[26], [27]. Carbon is well known to be very stable under thermal stress, but, its electrical stability under current stress is not so clear. Current stability was measured at room temperature under the common-base configuration to keep the emitter current constant at 10 mA. In the Gummel plot, the current shift toward higher base-emitter voltage often observed for such samples is attributed to the drift of the junction toward the wide-gap AlGaAs layer due to Be diffusion. In contrast, a GSMBE-grown carbon-doped base HBT with an emitter size of 5×5 μm^2 shows no change in the Gummel plot even after 10 hours of current stress. This indicates the absence of base dopant diffusion due to electric stress.

The collector current, I_c, and base current, I_b, of the carbon-doped-base HBT, were measured as a function of stress time with the emitter current kept constant at 10 mA, which is the operation current commonly used for practical applications. The variation of I_c/I_b with current stress time, which expresses the variation in gain, is shown in Fig. 12. The gain was almost constant over the investigated duration of up to

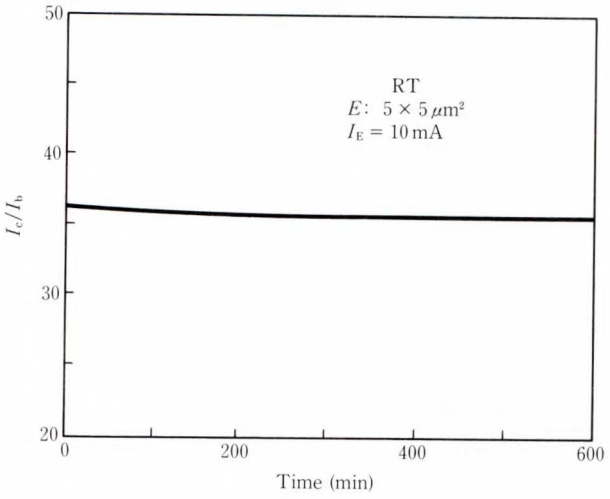


Fig. 12 – Variation of gain (I_c/I_b) with electrical stress time.

10 hours, suggesting no significant degradation of device performance for GSMBE-grown carbon-doped base HBT. These results suggest that carbon is stable under thermal stress during growth and under current stress.

4. Conclusion

We compared the incorporation of oxygen and carbon background impurities into AlGaAs and the n-type doping of AlGaAs with uncracked Si_2H_6 as an n-type dopant source, using TEAl and TMAAl as aluminum sources for GSMBE using only gaseous sources. It was shown that carrier concentrations in n-AlGaAs can be well controlled around 10^{18} cm^{-3} using TEAl and from 2×10^{17} cm^{-3} to 3×10^{18} cm^{-3} using TMAAl. The GaAs epilayer hole concentration could be easily controlled between 6.3×10^{18} and 1.3×10^{20} cm^{-3} by varying V/III ratio. We grew a carbon-doped base ($p = 4 \times 10^{19}$ cm^{-3}, 92.5-nm thick, TMGa as the carbon source) GaAs/AlGaAs HBT having a silicon doped emitter layer ($n = 9 \times 10^{17}$ cm^{-3}, TEAl as the aluminum source). The emitter-base ideality factor was 1.12, indicating that an excellent junction was formed using silicon and carbon. The dc current gain was 45 at a current density of 4×10^4 A/cm² (4×5 μm^2 emitter). Device characteristics under current stress was found to be stable. These results demonstrate the great potential of GSMBE for HBT applications.

References

1) Yamada, T., Tokumitsu, E., Saito, K., Akatsuka, T., Miyauchi, M., Konagai, M., and Takahashi, K.: Heavily Carbon Doped p-Type GaAs and GaAlAs Grown by Metalorganic Molecular Beam Epitaxy. *J. Crystal Growth*, **95**, 1-4, pp. 145-149 (1989).

2) Saito, K., Tokumitsu, E., Akatsuka,T., Miyauchi, M., Yamada, T., Konagai, M., and Takahashi, K.: Characterization of p-Type GaAs Heavily Doped with Carbon Grown by Metalorganic Molecular-Beam Epitaxy. *J. Appl. Phys.*, **64**, 8, pp. 3975-3979 (1988).

3) Ren, F., Abernathy, C. R., Pearton, S. J., Fullowan, T. R., Lothian, J., and Jordan, A. S.: GaAs-AlGaAs HBT with Carbon Doped Base Layer Grown by MOMBE. *Electron. Lett.*, **26**, 11, pp. 724-725 (1990).

4) Sandhu, A., Fujii, T., Ando, H., Takahashi, T., Ishikawa, H., Okamoto, N., and Yokoyama, N.: Gas Source MBE Growth GaAs/AlGaAs Heterostructure Bipolar Transistor with a Carbon Doped Base Using Only Gaseous Sources. *Jpn. J. Appl. Phys.*, **30**, 3, pp. 464-465 (1991).

5) Ando, H., Sandhu, A., Ishikawa, H., Sugiyama, Y., and Fujii, T.: Gas-Source MBE Growth of AlGaAs Using TEG, TEA and AsH_3. Proc. 16th Int. Symp. Gallium Arsenide and Related Compounds, Inst. Phys. Conf. Ser. 106, Karuizawa, 1989, Ikoma, T. and Watanabe, H., eds. Bristol and New York, Inst. Phys., 1990, pp. 217-222.

6) Skevington, P. J., Andrews, D. A., and Davies, G. J.: Anomalous Silicon and Tin Doping Behavior in Indium Phosphide Grown by Chemical Beam Epitaxy. *Appl. Phys. Lett.*, **56**, 16, pp. 1546-1548 (1990).

7) Sandhu, A., Fujii, T., Ando, H., and Ishikawa, H.: A Study of Cold Dopant Sources for Gas Source MBE: the Use of Disilane as an n-Type Dopant of $Al_xGa_{1-x}As$ (x = 0-0.28) and Trimethylgallium as a p-Type Dopant of GaAs. *Jpn. J. Appl. Phys.*, **29**, 7, pp. L1033-L1035 (1990).

8) Fujii, T., Sandhu, A., Ando, H., Kataoka, Y., and Ishikawa, H.: Doping Characteristics of

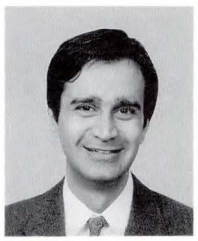
Gas-Source MBE-Grown n-$Al_xGa_{1-x}As$ (x = 0-0.28) Doped Using Disilane. *Jpn. J. Appl. Phys.*, **29**, 11, pp. 2386-2387 (1990).

9) Abernathy, C. R., Jordan, A. S., Pearton, S. J., Hobson, W. S., Bohling, D. A., and Muhr, G. T.: Growth of High Quality AlGaAs by Metal-organic Molecular Beam Epitaxy Using Trimethylamine Alane. *Appl. Phys. Lett.*, **56**, 26, pp. 2654-2656 (1990).
10) Okamoto, N., Ando, H., Sandhu, A., and Fujii, T.: Gas Source Molecular Beam Epitaxy Growth of High Quality AlGaAs Using Trimethylamine Alane as the aluminum Source. *Jpn. J. Appl. Phys.*, **30**, 12B, pp. 3792-3795 (1991).
11) Lee, B. J., Houng, Y. M., Miller, J. N., and Turner, J. E.: Carbon Incorporation in AlGaAs Grown by CBE. *J. Crystal Growth*, **105**, pp. 168-177 (1990).
12) Hersee, S. D., Martin, P. A., Chin, A., and Ballingall, J. M.: The Growth of High- Quality AlGaAs Metalorganic Molecular- Beam Epitaxy. *J. Appl. Phys.*, **70**, 2, pp. 973-976 (1991).
13) Fujii, T., Okamoto, N., Sandhu, A., Ando, H., and Kataoka, Y.: Si Doping of AlGaAs Using Si_2H_6 Grown by Gas-Source Molecular-Beam Epitaxy Using Trimethylamine Alane, Triethyl-gallium, and AsH_3. *J. Vac. Sci. Technol.*, **B10**, 2, pp. 946-948 (1992).
14) Takahashi, T., Yamaguchi, Y., Sandhu, A., Ando, H., Fujii, T., and Yokoyama, N.: Carbon-Doped-Base AlGaAs/GaAs HBTs Grown by Gas-Source Molecular Beam Epitaxy Using Only Gaseous Sources. *Jpn. J. Appl. Phys.*, **30**, 12B, pp. 3843-3845 (1991).
15) Ishikawa, H., Ando, H., Kondo, K, Sandhu, A., Miyauchi, E., Fujii, T., and Hiyamizu, S.: Metalorganic Gas Control System for Gas Souce Molecular Beam Epitaxy. *J. Vac. Sci. Technol.* **A8**, 2, pp. 805-810 (1990).
16) Benchimol, J. L., Alexandre, F., Gao, Y., and Alaoui, F.: Growth Parameter Dependence of Background Doping Level in GaAs, $In_{0.53}Ga_{0.47}As$ and $Al_xGa_{1-x}As$ Grown by Metalorganic Molecular Beam Epitaxy. *J. Crystal Growth*, **95**, 1-4, pp. 150-153 (1989).
17) Putz, N., Heinecke, H., Heyen, M., Balk, P., Weyers, M., and Luth, H.: A Comparative Study of $Ga(CH_3)_3$ and $Ga(C_2H_5)_3$ in the MOMBE of GaAs. *J. Crystal Growth*, **74**, pp. 292-300 (1986).
18) Chiu, T. H., Tsang, W. T., Schubert, E. F., and Agyekum, E.: Chemical Beam Epitaxial Growth of High-Purity GaAs Using Triethylgallium and Arsine. *Appl. Phys. Lett.*, **51**, 14, pp. 1109-1111 (1987).
19) Abernathy, C. R., Pearton, S. J., Baiocchi, F. A., Ambrose, T., Jordan, A. S., Bohling, D. A., and Muhr, G. T.: Effect of Source Chemistry and Growth Parameters on AlGaAs Grown by metalorganic Molecular Beam Epitaxy. *J. Cryst. Growth*, **110**, pp. 457-471 (1991).
20) Abernathy, C. R., Pearton, S. J., Bohling, D. A., and Muhr, G. T.,: The Roles of Aluminum and Hydrogen in Impurity Contamination of AlGaAs Grown by MOMBE. *J. Crystal Growth*, **111**, pp. 574-577 (1991).
21) Heinecke, H., Werner, K., Weyers, M., Luth, H., and Balk, P.: Doping of GaAs in Metalorganic MBE Using Gaseous Sources. *J. Crystal Growth*, **81**, pp. 270-275 (1987).
22) Kuech, T. F., Veuhoff, E., and Meyerson, B. S.: Silicon Doping of GaAs and $Al_xGa_{1-x}As$ Using Disilane in Metalorganic Chemical Vapor Deposition. *J. Crystal Growth*, **68**, pp. 48-53 (1984).
23) Weyers, M., Putz, N., Heinecke, H., Heyen, M., Luth, H., and Balk, P.: Intentional p-Type Doping by Carbon in Metalorganic MBE of GaAs. *J. Electron. Mat.* **15**, 2, pp. 57-59 (1986).
24) Ando, H., Fujii, T., Sandhu, A., Takahashi, T., Ishikawa, H., Okamoto, N., and Yokoyama, N.: Growth of Carbon-Doped Base GaAs/AlGaAs HBT by Gas-Source MBE Using TEG, TEA, TMG, AsH_3, and Si_2H_6. Proc. 3nd Int. Conf. Chemical Beam Epitaxy and Related Growth Techniques, Oxford, 1991, *J. Crystal Growth*, **120**, pp. 228-233 (1992).
25) Ren, F., Fullowan, T. R., Abernathy, C. R., Pearton, S. J., Smith, P. R., Kopf, R. F., Laskwski, E. J., and Lothian, J. R.: Selfaligned AlGaAs/GaAs HBT Grown by MOMBE. *Electron. Lett.*, **27**, 12, pp. 1054-1056 (1991).
26) Hafizi, M. E., Pawlowicz, L. M., Tran, L. T., Umemoto, D. K., Streit, D. C., Oki, A. K., Kim, M. E., and Yen, K. H.: Reliability Analysis of GaAs/AlGaAs HBT's Under Forward Current/ Temperature Stress. GaAs IC Symp. Tech. Dig., IEEE, 1990, pp.329-332.

27) Nakajima, O., Ito, H., Nittino, T., and Nagata, K.: Current Induced Degradation of Be-Doped AlGaAs/GaAs HBT's and Its Suppression by Zn Diffusion into Extrinsic Base Layer. IEDM 90, San Francisco, 1990, pp. 673-676.

Toshio Fujii received the B.S. and Dr. degrees in engineering from Osaka University, Osaka, Japan, in 1971 and 1989.
He joined Fujitsu Laboratories Ltd. in 1971 and has been engaged in research and development of III - V compound semiconductors grown by molecular beam epitaxy for optical and high-speed devices.
He received a Paper Award in 1982 from the Japan Society of Applied Physics, and an Excellent Paper Award in 1993 from the Institute of Electronics, Information and Communication Engineers.

Hideyasu Ando received the B.S. degree in electrical engineering from Keio University, Kanagawa, Japan, in 1981, and M.S. and Dr. degrees in physical electronics from Tokyo Institute of Technology, Tokyo, Japan, in 1984 and 1987. He joined Fujitsu Laboratories Ltd., Atsugi, in 1987 and has been engaged in research and development of Gas-source MBE growth of III - V compound semiconductors for high-speed electron devices. He is a member of the Japan Society of Applied Physics.



Adarsh Sandhu received his Ph.D from Manchester University, England, UK. He joined Fujitsu Laboratories Ltd., Atsugi, in 1986 and has been engaged in research and development of III - V compound semiconductors grown by gas source MBE. He is a member of The Japan Society of Applied Physics (Japan) and The Institute of Physics (UK).

Naoya Okamoto received the B.S. and M.S. degrees in material science from Osaka University, Osaka, Japan, in 1988 and 1990.
He joined Fujitsu Laboratories Ltd., Atsugi, in 1990 and has been engaged in research and development of III - V compound semiconductors grown by Gas-source molecular beam epitaxy for high-speed devices such as heterojunction bipolar transistors.
He is a member of the Japan Society of Applied Physics.

UDC 621.395.74

Service Creation Environment Based on Application Oriented Specification Language

● Jun Maeda ● Moo Wan Kim ● Hideo Yunoki

(Manuscript received December 2, 1992)

This paper describes Fujitsu's research on an advanced service creation environment for intelligent networks. This paper introduces a service creation environment that enables rapid and easy development of new telecommunication services by personnel who are not experts in switching software. An application oriented service specification language based on a state transition model is proposed. An abstracted switching model and service independent processing components are used in this language to simplify specification description. This paper also discusses service development support functions (creation, verification, and translation of the specification) and outlines a prototype system.

1. Introduction

The use of advanced telecommunication services, for example, multimedia, intelligent control, and personalized communications, is expected to increase. These services are implemented by combining a telecommunication network with information processing functions by using computers connected to the network. Since service execution is controlled by software, the realization of these services involves a large amount of software design and development. Currently, service software is developed only by switching-software engineers of the telecommunications system vendors. However, for the rapid introduction of new services, carriers and subscribers (end users) must also be able to develop service software. To meet this requirement of "customer programmability", an improved service creation environment (SCE) is necessary[1]. Recently, a network architecture called the intelligent network (IN) has been introduced[2)-4)], and workstations are improving at a remarkable rate. The SCE is implemented on the workstations and creates service software for the IN.

Chapter 2 of this paper looks at the basic reasons and technical requirements for providing customer programmability. Chapters 3 and 4 describe an application oriented service specification language and service development support functions which enable flexible service development. Chapter 5 describes a prototype system based on the concepts and components described in the chapters above.

2. Service creation environment (SCE)

2.1 Intelligent network (IN)

The target network architecture of this research is the intelligent network (IN). The IN architecture makes it easier to develop service processing programs. Also, because service processing is separated from basic call processing, the IN architecture is expected to simplify and speed up the development of services. Figure 1 shows the architecture of the IN. It consists of three nodes: the service switching point (SSP), service control point (SCP), and service management system (SMS). The SSP performs the basic call processing that provides the ordinary switching functions. The

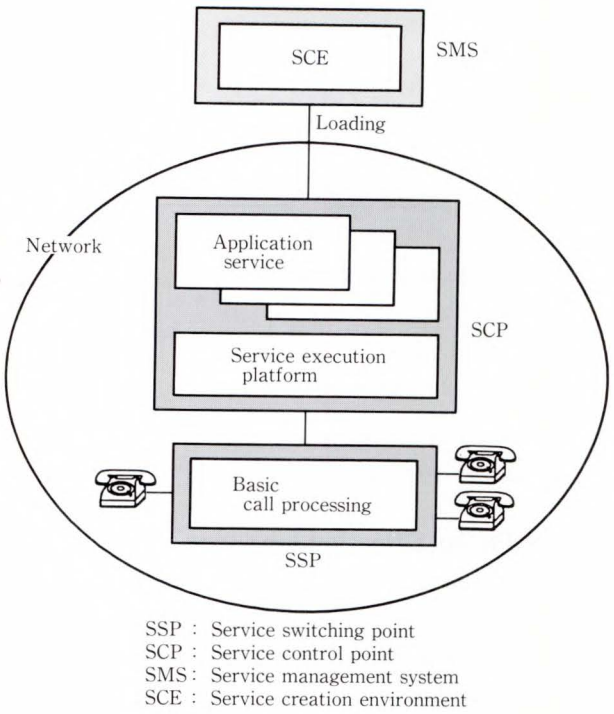


Fig. 1 – IN architecture.

SCP is a high performance, fault-tolerant computer that executes application service processing without affecting the basic call processing. The SSP and SCP communicate with each other via a standard message interface. The SMS manages information for service control. The creation, management, and operation of services are performed on the SMS. The services created by the SCE residing in SMS are downloaded to the SCP for execution.

2.2 Objective

Customer programmability is implemented in the data level, which is mainly used to define service parameters, and the logic level, which is the level used to create the service logic (program). Second level programming is difficult because it requires a knowledge of telecommunications and information processing. Although customer programmability in the data level has been developed, especially for free phone type services, it is still restricted. To enable the flexible developmemt of services, service logic creation is necessary. Also, when the situation involves a large amount of differing service requirements, service logic development should

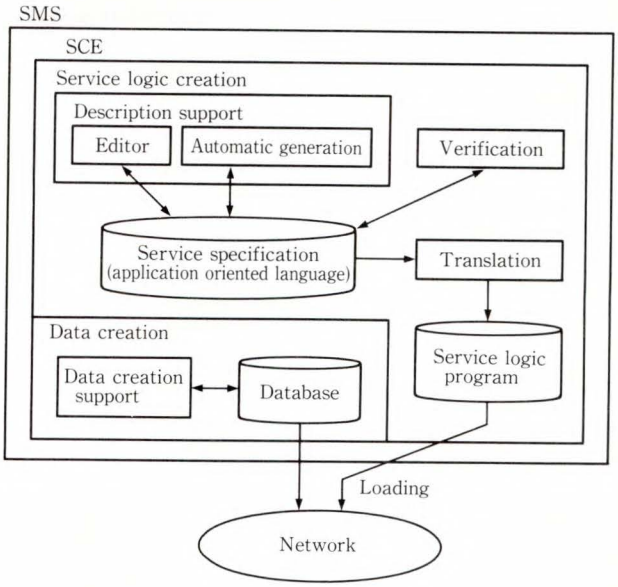


Fig. 2 – SCE configuration.

be performed not only by switching software experts but also by non-experts ; for example, the carrier's network operators and the computer-side's systems engineers. The objective of the SCE discussed in this paper is customer programmability, including service logic creation by non-experts.

Services are generally developed in the following stages:

1) planning,
2) preparation of the service specifications,
3) creation of related programs and data, and
4) installation of the program into the network.

Verification is performed at each stage. Although the SCE supports all of the development stages, this paper focuses on the service specification creation phase because it is regarded as the key issue affecting customer programmability.

2.3 Requirements

Figure 2 outlines the SCE configuration. The SCE performs two main functions: service logic creation and data creation.

The application oriented specification language is a key technology in service logic creation. Because users are non-expert, they cannot take into consideration the technical aspects of the service execution control logic.

Therefore, the user must be provided with a high level, easy-to-understand specification language. Users should be able to perform all service development tasks with only a knowledge of this language. It is also necessary to reduce the complexity of the service logic.

The service development process should be improved by using a rapid prototyping method that reduces the time required to find errors and complete the development. The service specification is first defined by using the service specification language. Then, the specification is verified, and if an error is found, the process is repeated. When the specification is completed, it is translated into an executable program.

To support this process, the following facilities are required:

1) Support for the description of the service logic specifications, including automatic generation of the specification.
2) Verification by testing at the service logic specification level.
3) Automatic translation of the service logic specification into an executable program.

Primitive data representations such as relational databases are not appropriate for non-expert users. Therefore, a higher level of data representation, that is, one that is conceptually matched to the user's view of IN service processing, should be provided for easy data creation. Such a representation must not be service-specific. Data is created by using data definition and data registration functions, and support facilities for both functions should be provided.

Since the service creation environment is an interactive system, the quality of the human interface is an important factor. Therefore, advanced interface facilities, for example, graphic and sound interfaces, should be used to provide an intuitive and user-friendly interface. Because users prefer differing types of interfaces, for example, some users prefer to enter data using an interactive method whereas others prefer to fill in a table, the users should be able to adapt the interface to suit their own preferences.

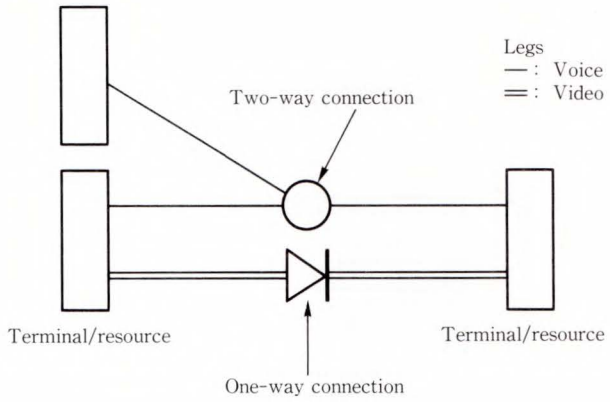


Fig. 3 – Example of call model state.

3. Service specification language

From the requirements described above, a formal language based on the state transition model is a suitable choice for the service specification language.

The behavior of the SSP can be regarded as a basic call model from the viewpoint of the SCP. This model is defined as a state transition representation. The service logic controls service execution along the basic call model. Therefore, the specification describes the service logic control flow based on the state transition. This description is well-known, easy-to-understand, and can be translated directly into an executable program. A formal language is needed for automatic translation[5].

The state transition model consists of the state and the state transition. The state represents the state of the call model, which is an abstracted representation of network capability. The call model simplifies the description by hiding the details of switching. Communication resources such as terminals, connection states, and connection paths are represented as communication subjects, connection points, or legs. To enable extension to a broadband ISDN, connection points have connection type attributes that specify whether one-way, two-way, or multi-way connections are possible. Communication subjects connect multiple legs, making multimedia service descriptions possible. Figure 3 shows an example state of the call model. The figure shows a two-way voice path between three communication subjects, and a one-way video path

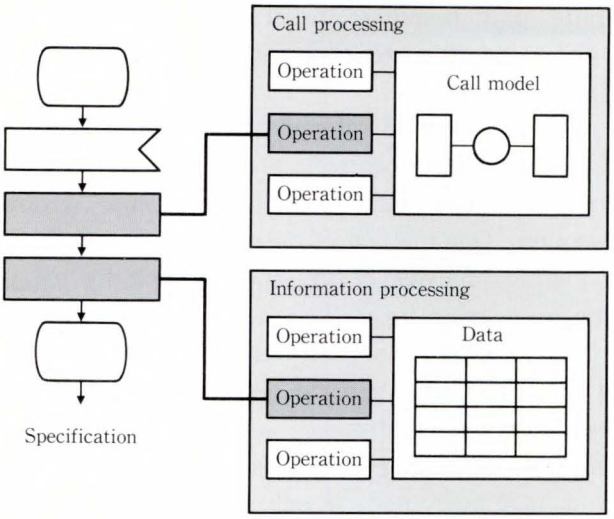


Fig. 4 – Service components.

between two of the communication subjects[6].

The state transition is represented by the service components, which are service independent processing elements[7]. The service components must be easy for the user to understand and must indicate simple, intuitive elements of service behavior.

Because of the need for modularity, the service components are based on an object-oriented concept. The objects consist of the call model, data, and related basic operations that are commonly used in many services (see Fig. 4). The service components are commands that invoke the objects. Some examples of these commands are connection and disconnection of legs, data authorization, and data translation.

The language should be represented in a textual or graphical form that is similar to SDL (specification and description language)[8]. A graphical representation of the call model and service logic sequence makes them easier for the user to understand.

4. Service development support functions

4.1 Creating the specification

In service development, after the target service feature has been determined, the designer creates the service specification. To simplify this creation stage, a specialized editor for the specification language should be available. An example of such an editor is a graphical editor that manipulates graphical representations of the specification language by using the graphical user interface (GUI) functions of workstations. Also, automatic creation of the service logic specification described below is required.

Because the formal service specification does not allow ambiguity, the service logic must be described in detail. This makes it difficult to write the specification. Moreover, in the case of multi-point, multi-connection services on a broadband ISDN, the complexity of the description grows combinatively. Therefore, the user cannot always be expected to write a complete service logic specification.

The specification consists of main and supplementary sequences, the user being mostly concerned with the main sequences. Supplementary sequences should be omitted because they are required in many parts of the specification, making it necessary to write many descriptions. To supplement the omitted specifications, a method of automatically generating them is needed. Because this is difficult to do algorithmically, knowledge processing technology should be used. The knowledge base contains information about how to make the service logic specification, and works as described below.

First, the knowledge base finds the sequence branch. (Many branches occur due to events or the execution of service components, and branches that are not described in the specification are taken into consideration.) The knowledge base contains information that indicates the important branches; for example, an on-hook event is usually important, and therefore usually requires a supplementary sequence.

Then, the knowledge base generates a processing plan. The states that the supplementary sequence should handle are determined from the cause of the supplementary sequence and the state of the call model. Because the processing usually results in one of several typical states, for example, path disconnection and error notification, it is feasible to determine the rules of the plan generation.

Finally, based on the processing plan, the knowledge base generates a processing sequence

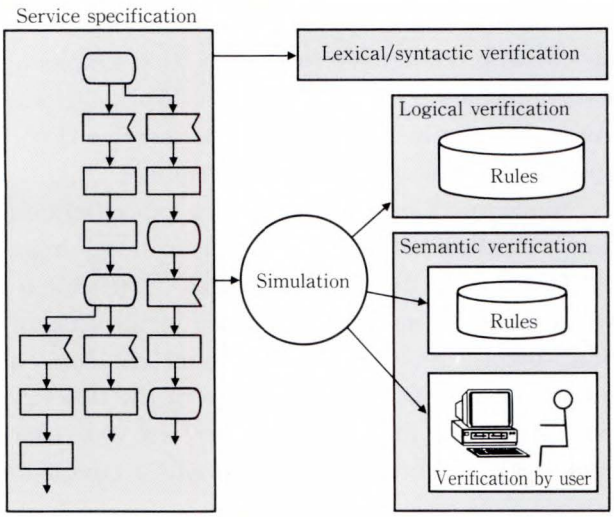


Fig. 5 – Methods of specification verification.

from the service components which change the state. The knowledge base selects the service components based on the results of service component execution. Then, the order of the service components is decided based on the constraints imposed on the order.

4.2 Verifying the specification

The verification function corrects errors in the specification. The verification[9] method that is used varies with the type of error being corrected. The methods of specification verification are shown in Fig. 5 and are as follows:

1) Lexical and syntactic

Lexical and syntactic errors are detected by using ordinary lexical and syntactic analysis. These errors include misspellings of service component names, incorrect parameter values, and errors in the sequence of service components.

2) Logical

This method detects various contradictions in the specification, for example, events that cannot occur in the specified state and service components that cannot be executed in the specified state. One of the verification methods is to run the specification on a simulation and then check the results by using verification rules contained in the knowledge base. The rules describe constraints imposed on service execution. The service execution can be regarded as a finite state machine; therefore, every action is verified based on reachability tree analysis.

3) Semantic

This method detects incorrect service behavior, for example, providing the wrong announcement and connecting the wrong terminal. One method of detecting these errors is semantic analysis of the specification by using the knowledge base. The results of the service simulation are checked using semantic rules which represent general principles of service behavior. Constructing a knowledge base involves collecting examples of semantic errors that have been made in actual specifications and then extracting rules from them. However, it should be noted that because the definition of semantic errors is sometimes ambiguous (definition depends on service designers), semantic analysis cannot always find every error. Another important method of detecting incorrect service behavior is the verification of the simulation output of service execution by user. To enable accurate evaluation of service use by using this method, it should be possible to run a simulation as if a real system was being operated.

4.3 Translation

The translation function translates the service logic program into a programming language, for example, C. An executable program is obtained by compiling this output. The service logic program consists of the message reception part and the service component execution part. The state of the call model and the event in the specification are mapped to the message reception part, and are used to specify the condition that the SSP sends the next event to the SCP. The service component is replaced by library programs that are invoked after message reception. The library programs include functional components (FCs) that are basic functions of the SCP platform. Some service components may have internal state transitions, in which case the service components are expanded into several parts.

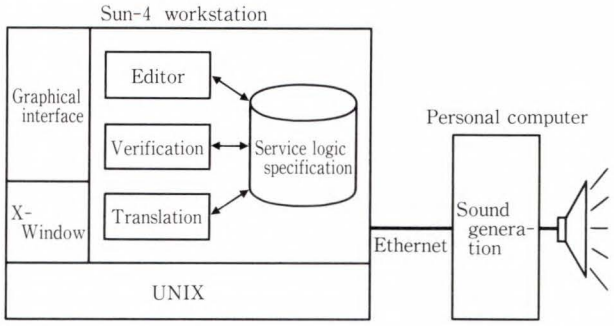


Fig. 6 – Configuration of prototype system.

5. Prototype system

5.1 System configuration

An SCE prototype system based on the concepts described above has been developed. The target switching system has a broadband ISDN switching capability based on ATM switching technology. Although the initial scope of this prototype system is limited to ordinary telephone services, an extension for broadband services is planned. The system consists of a Sun-4 workstation and a personal computer, which is used for sound generation. Figure 6 shows the configuration of the prototype system. This system uses the X-Window graphical user interface. The editing, verification, and translation programs are independent and access a common service specification database. These programs are written in C.

5.2 Specification language

The service logic specification language is represented in graphical form, and the specifications are written by a graphical editor. An example graphical specification is shown in Fig. 7. The specification shows the processing flow of service execution control. It consists of the call model states, the events and the service components. Textual representation is also available.

The state of the call model expresses the state of the telephone call, for example, Collecting Information, Analyzing Information, and Active (talking). Four terminal operations are provided as the events, for example, OFF-HOOK and ON-HOOK. Fifteen service components are provided. Examples of service components are shown in Table 1. The sequence can be branched according to the results of the service components.

Table 1. Examples of service components

Service component	Function
Announcement	Provides an announcement
Collect-user-information	Collects a number from the user
Merge	Combines two connection points and generates a conference call state
Remove	Removes a leg
Translation	Translates a number using a database

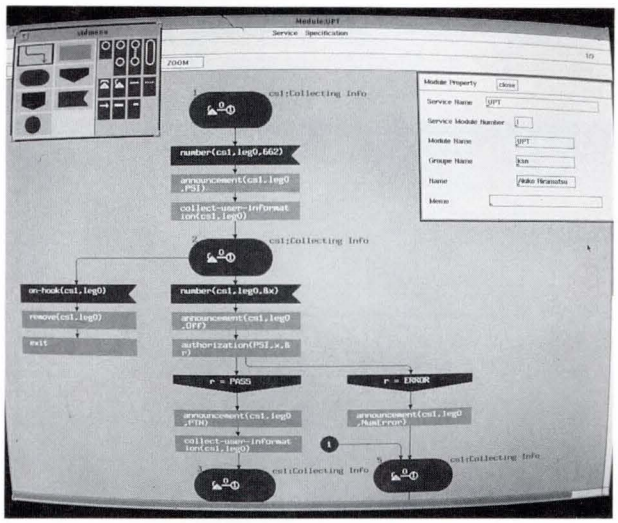


Fig. 7 – Graphical specification example.

For example, in Fig. 7, a trigger indicating that special number "662" has been dialed from a terminal is received in the Collecting Information state, and then the service is activated. The service specification instructs the service to output an announcement to the terminal and to wait for additional information from the terminal by using the service components. In the next Collecting Information state, service execution continues when the information is entered from the terminal. When the terminal is hooked on, an exceptional sequence is executed and the service is canceled. It is also possible to express a conference call by combining the state of the

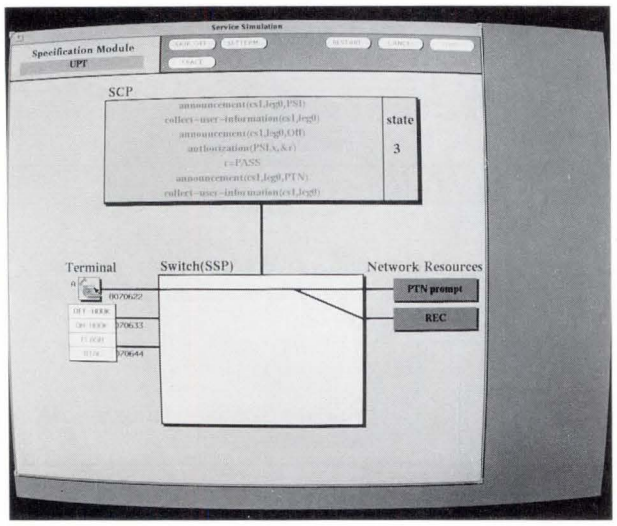


Fig. 8 – Verification example (1).

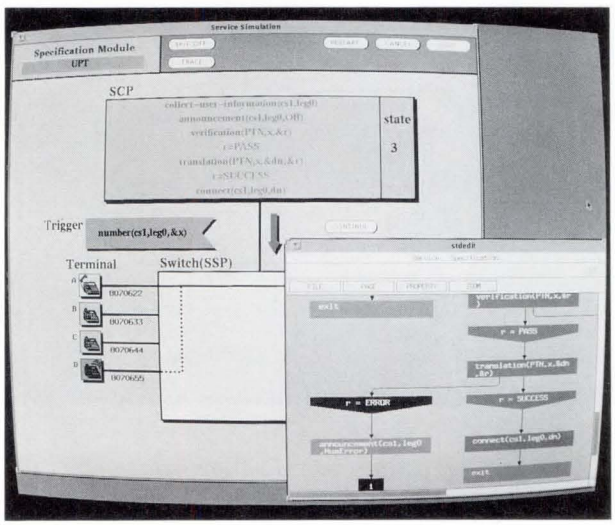


Fig. 9 – Verification example (2).

call model and the service components. This is done by creating a new connection point for the third called party and merging it with the existing connection point.

5.3 Service simulation

A semantic verification tool that runs the service logic specification under a simulation was developed. The simulation demonstrates how the service is executed on the screen. Also, the simulation outputs various tones and announcements to terminals to explain the service execution. The simulation is operated directly by specifying terminal icons and

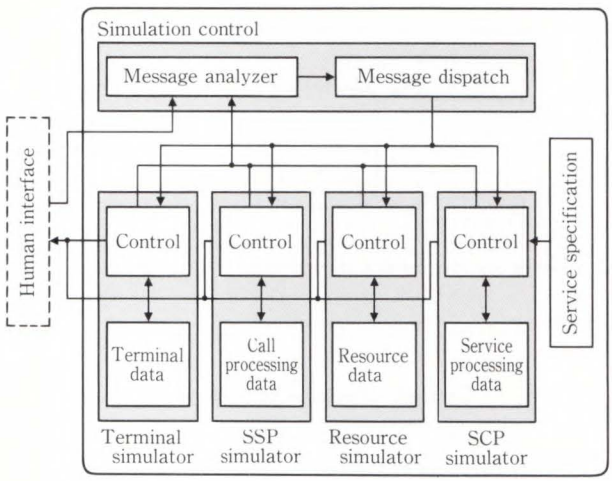


Fig. 10 – Simulation mechanism.

commands in menus such as OFF-HOOK, ON-HOOK, and DIAL. In this simulation, the service can be executed in steps. The user can easily find errors by using a function that shows the service execution trace on the specification. Some examples of verification screens are shown in Figs. 8 and 9. The connection state is shown in the switch (SSP) box and the executed service components are displayed in the SCP box. In Fig. 9 the trace of the execution route is displayed in a window on the right.

Using this tool to repeat the verification and refinement stages of specification creation ensures sufficient verification in the semantic level, and enables services to be developed efficiently. Because large portions of the specifications for new services are identical or similar to existing specifications, reuse of existing specifications is encouraged. This makes this simulation even more useful because it can help the user to understand the behavior of an existing specification.

The simulation mechanism is shown in Fig. 10. The simulators for the terminals, network resources, SSP, and SCP are independent and communicate using messages. When the SCP simulator receives a trigger or event message from the SSP simulator, it executes the service specification by using an interpreter and sends control messages to the SSP simulator. Simulation control integrates the simulators by switch-ing the messages.

5.4 Translation

The translation tool translates the service specification written in the service logic specification language into a service logic program. A service logic specification can be divided into blocks that start from a message waiting state and end at the next message waiting state. First, the translation tool generates the framework of the program that contains the variable declaration and the message receiving function. Then, the blocks are translated in the order in which they appear in the specification. The translation procedure for each block is as follows:

1) The conditional branches which depend on the message contents are generated.
2) The service components between a message waiting state and the next message waiting state are replaced with library programs that include FCs by applying the service component expansion rules.

5.5 Results

Two services were created on the prototype system so that the system could be evaluated. These two services were a simple universal personal telecommunications (UPT) service to represent a typical database-oriented service, and a conference service to represent a call processing-oriented service. The specifications for these two services were written using the specification editor, and the design was verified using the verification tool. The evaluation results indicated that the method worked well.

6. Conclusion

A service creation environment that provides customer programmability improves the development of intelligent network services. This paper presented an architecture that enables non-expert designers of telecommunication software to develop service logic. We propose an application oriented specification language which uses an abstracted call model and service components. Service development is performed at the specification level by using support facilities to create and verify the specification, and then translate it into an executable program. Further developments leading to more sophisticated services in an advanced telecommunication environment are expected.

References

1) Maeda, J., Kim, M. W., Hiramatsu, A., and Yunoki, H.: Service creation environment for the intelligent network. Proc. IEE SETSS'92, 1992, pp. 32-36.
2) CCITT: " Principles of intelligent network architecture". Draft recommendation Q.1201, 1992.
3) Bellcore: Advanced intelligent network (AIN) release 1 switching system generic requirements. TA-NWT-001123, 1991.
4) Sera, A., Matsuda, T., and Fujiyama, Y.: Enhancement of digital switching system FETEX-150 towards broadband ISDN and intelligent network. *FUJITSU Sci. Tech. J.*, **28**, 2, pp. 150-160. 1992.
5) Hasui, K., Miyazaki, K., Shimoji, Y., Kim, M. W., and Suzuki, T.: An in- telligent switching software development system – AIFLS and its processor. Proc. GLOBECOM'84, 1984, pp. 152-156.
6) Fukazawa, M., Wakamoto, M., and Kim, M. W.: Intelligent network call model for broadband ISDN. Proc. ICC'91, 1991, pp. 964-968.
7) CCITT: " Global functional plane for intelligent network CS-1". Draft recommendation Q.1213, 1992.
8) CCITT: "Specification and description language (SDL)". Recommendation Z.100, 1992.
9) Miyazaki, K., Yamazaki, J., Kakemizu, M., Iwami, Y., Kishida T., and Suzuki, T.: SVEX: switching program verification expert system. Proc. IEE SETSS'89, 1989, pp. 178-182.

Jun Maeda received the B.E. and M.E. degrees in electronics from University of Electro-Communications, Tokyo, Japan, in 1980 and 1982.
He joined Fujitsu Laboratories Ltd., Kawasaki, in 1982 and has been engaged in research and development of communication software and intelligent network.

Moo Wan Kim received the B.E. and Dr. degrees in electronics from Osaka University, Osaka, Japan, in 1974 and 1980.
He joined Fujitsu Laboratories Ltd., Kawasaki, in 1980 and has been engaged in research and development of communication software and intelligent network.

Hideo Yunoki received the B.E. degree in electronics from Waseda University, Tokyo, Japan, in 1975.
He joined Fujitsu Limited, Kawasaki, in 1975 and has been engaged in research and development of ISDN and Intelligent network of overseas switching System.

UDC 681.32:781.1

Interactive Music Composer Based on Neural Networks

● Masako Nishijima ● Kazuyuki Watanabe

(Manuscript received December 2, 1992)

Neuro-Musician is an interactive music composer which plays a jam session with a human pianist, while composing improvisations using neural networks. It employs a new method of handling musical contexts to play several measures of a melody. A melody generation model is invented for this purpose. This model is based on neural networks, and utilizes the following information: melody contours, pitch and duration, and note-on timing. Three kinds of networks are used to implement the melody generation model. This paper presents a description of the model, its mechanism, and an evaluation.

1. Introduction

A computer with musical sense can help a person make music. Given a phrase, for example, such a computer composes appropriate example of phrases that follow or substitute for it. These examples are helpful for people wishing to compose or play music.

Neuro-Musician is a computer with musical sense. It acquires its musical sense by learning a musical style using neural networks. Musical styles are difficult to describe with rules. One of the best ways for a computer to acquire a musical style is to give it actual music. We attempted to teach instances of music to neural networks.

2. Neural network

A neural network can be considered as a simple model of the human brain. The neural network we used is a three-layer hierarchical network that consists of an input layer, a hidden layer, and an output layer (see Fig. 1). Each layer has several neuron units. The network is taught pairs of input data and corresponding desired output data by the error backpropagation method[1]. This teaching fixes the strength of connection between nodes in the network, which determine its behavior. The network can then calculate an output for any input. For example, if the network is given an input which was used during teaching, it generates the output it was taught. As the neural network is taught pairs of inputs and desired outputs, it acquires generalized relationships between them. If the network is given a new input that was not used during teaching, it generates an appropriate output using these generalized relationships.

3. History

The "Neuro-Drummer" was our first research project. It ran from 1988 to 1989[2)-4)]. We attempted to teach a sense of rhythm to a neural network, because rhythm is one aspect of musical style. After the neural network had

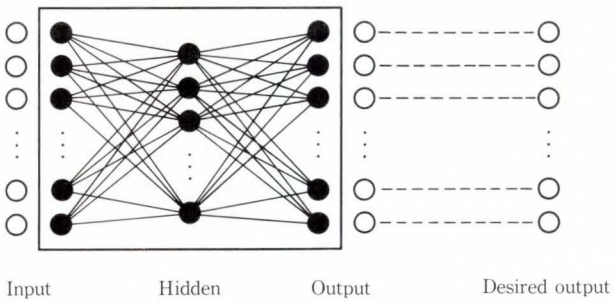


Fig. 1 – Neural network.

learned about forty pairs of an input rhythm pattern and an output rhythm pattern, a professional drummer conceded that the Neuro-Drummer had improved greatly, and that it usually replied with interesting rhythms.

The "Neuro-Musician" was our next research project[5]. During this project, we taught a sense of melody to neural networks. Learning this sense was more difficult, because it involved a lot of factors including pitch, duration, harmony, and rhythm, and because these factors influence each other.

The Neuro-Musician takes the place of one musician in an *ad-lib* session in which two players take turns playing. The first player plays a piece of music for several measures, then the second player, the Neuro-Musician replies to it. Each player needs to accept and adapt to the other player's musical style. Neuro-Musician's replies cannot be random, and they must make musical sense and be artistically satisfying.

4. Melody generation model

We investigated how a professional jazz musician approaches a jazz *ad-lib* session, and what factors are crucial in playing a phrase eight or sixteen measures long. The musician we interviewed listed the following four major factors:

1) Contour (outline) of the melody.
2) Pitch change and rhythm (These are used to compose a melody that satisfies the contour).
3) Note-on timing.
4) Chord progression and available note scales.

When the musician plays an *ad-lib* session, he or she considers all these factors together. For each piece of music, chord progression and available note scales can be determined. Relationships between the first three factors (contour, pitch and rhythm, and note-on timing) are determined when a musician plays phrases. The musician does not determine these relationships logically, but unconsciously. A musician will naturally play example phrases when he or she explains these relationships.

We made a melody generation model based on the factors above and our discussion with the jazz musician. Our melody generation model is an algorithm that lets a computer process music examples like a human musician would. We pay close attention to these three loose relationships: the relationship between the contour of a melody and the contour of the music immediately following the melody, the relationship between the contour of a melody and its detailed information such as pitch change and rhythm, and the relationship between the rhythm pattern and the note-on timing. Three factors are sampled as follows:

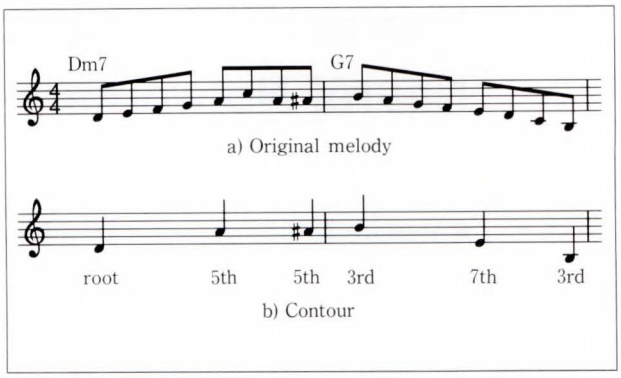


Fig. 2 – Contour.

1) *Contour* is described by the first, middle, and final notes in each measure using chord construction. In Fig. 2, first measure contour data consists of the root note, the fifth note, and the fifth note of D minor seventh (Dm7) instead of D, A, and A#, respectively.
2) *Pitch* means the difference between two consecutive notes. *Rhythm* is in units of a sixteenth of a note with strength (velocity). A number of notes and the distribution of pitch are also sampled from the melody. We call a pair of these two elements *note density*.
3) *Note-on timing* is the sampled difference between punctual playing to a musical score and real playing in units of the MIDI (Musical Instrument Digital Interface) timing clock.

5. Composing mechanism

The composing mechanism consists of three procedures (see Fig. 3) :

1) Make a contour (outline) of the output

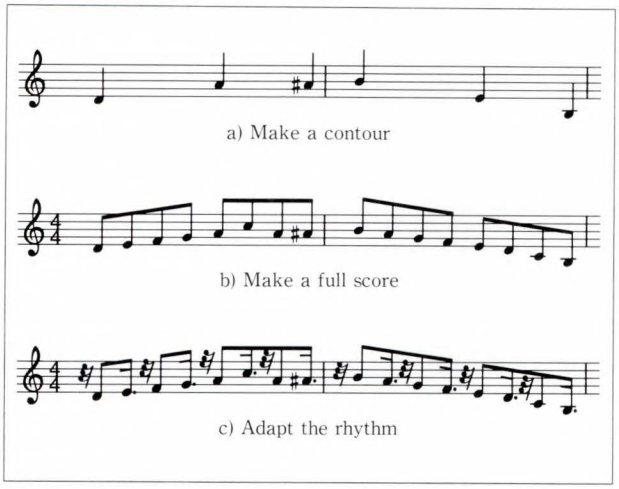


Fig. 3 – Composing mechanism.

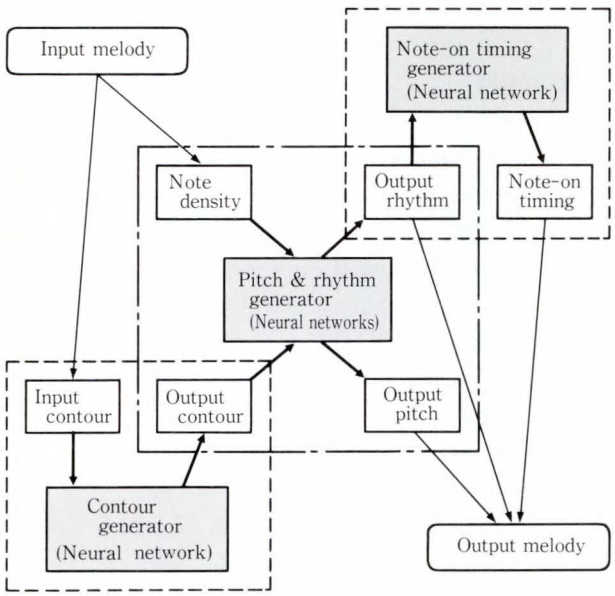


Fig. 4 – Melody generation model.

melody.

2) Make a full score by giving details to the contour.
3) Adapt the rhythm for jazz playing.

Three kinds of networks are used to implement our mechanism, and these networks cooperate to generate an output melody (see Fig. 4).

In the first step, a contour of an output melody is generated using a contour of an input melody. This is because most human pianists tend to grasp the contour first when playing jazz *ad-lib* sessions. A network (contour generator) is taught pairs of an input melody contour and a

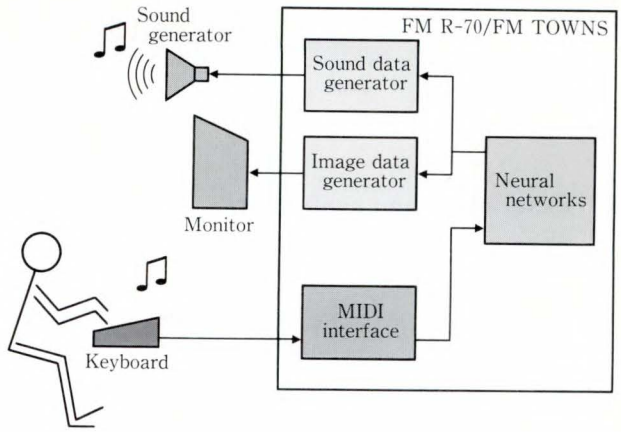


Fig. 5 – System configuration.

desired output melody contour. This network consists of 48 units in the input layer, 60 in the hidden layer, and 48 in the output layer.

In the second step, a full score for eight measures is made using the note density. A network (pitch and rhythm generator) is taught pairs of contour, note density, pitch, and rhythm (duration and accent). The network consists of four sub-networks. Each sub-network generates two-measure melody data and has 24 units in the input layer, 40 in the hidden layer and 32 in the output layer.

In the final step, swing is added to the rhythm to adapt it for jazz playing, which is essential to jazz. A network (note-on timing generator) is taught pairs of rhythm patterns and note-on timing. This network has 8 units in the input layer, 8 in the hidden layer, and 13 in the output layer.

6. System configuration

We ran this system on an FM R-70 32-bit personal computer and used MIDI to connect instruments to the computer. When a human musician plays an eight-measure melody, MIDI signals are generated and transmitted to the computer. The MIDI signals are converted to input data for the neural networks. The output from the neural networks is converted back into MIDI signals and these signals are transmitted to the sound generator to produce music (see Fig. 5). These MIDI signals are also transmitted to an FM TOWNS to synchronize computer

graphics on the FM TOWNS's monitor with Neuro-Musician's music. Prerecorded music for bass and drums are synchronized with the system.

7. Experiment and evaluation

We experimented by having Neuro-Musician play a jam session with a jazz pianist. To compose improvisations, the neural networks had to learn approximately 30 eight-measure patterns from the musician. This experiment was an eight-measure trade, and the theme was "Satin Doll" by Duke Ellington. When the human pianist plays eight measures of a melody, the Neuro-Musician replies with eight measures.

In this experiment, we were able to get very exciting "jam sessions" between the jazz pianist and the Neuro-Musician. We determined that the output from neural networks can be effective in playing a jam session. The Neuro-Musician plays responses that the human musician likes. This experiment did reveal, however, an unexpected constraint on the music – unnatural replies came out, and due to several reasons, the computer did not generate human-like music. The Neuro-Musician generates arbitrarily long or fast phrases, either without rests or filled with sixteenth notes. This results in music that clearly would not be played by a human player. Another point is that the note-on timing patterns tend to be too simple. This kind of music rarely pleases or excites listeners. We must teach our neural networks additional human characteristics, such as limits on hand movement and breathing intervals. It is important, for example, to consider a relationship between the length of phrases in the performance and breathing intervals.

8. Conclusion

An interactive music composer was created by investigating the musical factors involved in an *ad-lib* jazz session with a human musician. Three kinds of neural networks were able to generalize the relationships between musical factors by learning instances of music. In the future, we are going to refine our melody generation model by adding more human characteristics.

9. Acknowledgment

We would like to thank Takashi Miyazawa and Jun-ichi Tamaru for their performances.

References

1) Rumelhart D. E., Hinton G. E. and Williams R. J.: "Learning Internal Representations by Error Propagation." Parallel Distributed Processing, **Vol.I**, MIT Press, 1986, pp. 318-364.
2) Nishijima M. and Kijima Y.: Learning on Sense of Rhythm with a Neural Network – THE NEURO DRUMMER –. Proc. 1st Int. Conf. Music Perception and Cognition, 1989, pp. 77-80.
3) Rheingold H.: "From Neuro-Drummers to First-Person Fantasies". VIRTUAL REALITY, SUMMIT BOOKS, 1991, pp. 294-299.
4) Nishijima M., and Murakami K.: The Neuro-Drummer. (In Japanese), *J. Soc. Instr. & Control Engrs.*, **30**, 4, pp. 344-347 (1991).
5) RETI NEURALI: FUJITSU NEURO-MUSICIAN. (In Italian), *STRUMENTI MUSICALI*, WINTER NAMM'92, pp. 54-59 (1992).

Masako Nishijima received the B.S. and M.S. degrees in computer science from Tokyo University of Agriculture and Technology in 1984 and 1986. She joined Fujitsu Laboratories Ltd., Kawasaki, in 1986 and has been engaged on research in computer music since 1988.
She is a member of International Computer Music Association and Information Processing Society of Japan.

Kazuyuki Watanabe received the B.E. and M.E. degrees in nuclear engineering from Hokkaido University in 1988 and 1990. He joined Fujitsu Laboratories Ltd., Kawasaki, in 1990 and has been engaged on research in computer music.
He is a member of Information Processing Society of Japan (IPSJ), and received IPSJ Convention Award in 1992.

International Network

Offices

Fujitsu Abu Dhabi
P.O. Box 47047 Suite 802,
A1 Masadood Tower,
Sheikh Hamdan Street,
Abu Dhabi, U.A.E.
Telephone : (971-2)-333440
FAX : (971-2)-333436

Amman Project Office
P.O. Box 5420, Ammán, Jordan
Telephone : (962)-6-662417
FAX : (962)-6-673275

Bangkok Office
3rd Floor, Dusit
Thani Bldg., 1-3, Rama IV,
Bangkok10500, Thailand
Telephone : (66-2)-236-7930
FAX : (66-2)-238-3666

Beijing Office
Room 2101, Fortune Building,
5 Dong San Huan Bei-Lu,
Chao Yang District, Beijing,
People's Republic of China
Telephone : (86-1)-501-3261
FAX : (86-1)-501-3260

Brussels Office
Avenue Louise 176, Bte 2
1050 Brussels, Belgium
Telephone : (32-2)-648-7622
FAX : (32-2)-648-6876

Harare Office
2nd Floor, CABS House,
Corner Central Avenue,
4th Street, Harare, Republic of Zimbabwe
Telephone : (263-4)-732-627
FAX : (263-4)-732-628

Hawaii Office
6660 Hawaii Kai Drive, Honolulu,
Hawaii 96825, U.S.A.
Telephone : (1-808)-395-2314
FAX : (1-808)-396-7111

Jakarta Representative Office
16th Floor, Skyline Bldg.,
Jalan M.H. Thamrin No.9,
Jakarta, Indonesia
Telephone : (62-21)-333245
FAX : (62-21)-327904

Kuala Lumpur Liaison Office
Letter Box No.47,
22nd Floor, UBN Tower No.10,
Jalan P. Ramlee, 50250,
Kuala Lumpur, Malaysia
Telephone:(60-3)-238-4870
FAX : (60-3)-238-4869

Munich Office
c/o Siemens Nixdorf Informationsysteme
AG, D8 SC,
Otto-Hahn Ring 6, D-8000,
München 83, Germany
Telephone:(49-89)-636-3244
FAX : (49-89)-636-45345

New Delhi Liaison Office
Mercantile Home
1st Floor, 15 Katsurba, Gandhi Marg
New Delhi 110001, India
Telephone : (91-11)-331-1311
FAX : (91-11)-332-1321

New York Office
680 Fifth Avenue, New York,
N.Y. 10019, U.S.A.
Telephone : (1-212)-265-5360
FAX : (1-212)-541-9071

Oficina Informativa en Mexico
Paseo de la Reforma No.255 Oficina 13-A
Col. Cuauhtemoc 06500, Mexico, P. F.
Telephone : (525)-592-3940
FAX : (525)-592-3985

Shanghai Office
Room 705, West Podium Office Building,
Shanghai Centre,
1376 West Nanjing Road,
Shanghai 200040, People's Republic of China
Telephone : (86-21)-279-8410
FAX : (86-21)-279-8411

Sucursal de Colombia
Carrera. 9a. A No. 99-02 Ofc. 811
Edificio Seguros Del Comercio
Santa Fe De Bogotá, Colombia
Telephone : (57-1)-618-3147
FAX : (57-1)-618-3134

Taipei Office
Sunglow Bldg., 8F 66, Sung Chiang
Road, Taipei, Taiwan
Telephone : (886-2)-551-0233
FAX : (886-2)-536-7454

Washington, D.C. Office
1776 Eye Street, N.W.,
Suite 880, Washington, D.C.,
20006, U.S.A.
Telephone : (1-202)-331-8750
FAX : (1-202)-331-8797

Subsidiaries

ASIA AND OCEANIA

Fujitsu Australia Ltd.
Fujitsu Australia Software Technology, Pty Ltd.
Fujitsu Component (Malaysia) Sdn. Bhd.
Fujitsu Hong Kong Ltd.
Fujitsu Korea Ltd.
Fujitsu Microelectronics Asia Pte. Ltd.
Fujitsu Microelectronics (Malaysia) Sdn. Bhd.
Fujitsu Microelectronics Pacific Asia Ltd.
Fujitsu New Zealand Ltd.
Fujitsu (Singapore) Pte. Ltd.
Fujitsu (Thailand) Co., Ltd.
Fujitsu Trading Ltd.

NORTH AMERICA

Fujitsu America, Inc.
Fujitsu Business Communication Systems, Inc.
Fujitsu Canada, Inc.
Fujitsu Compound Semiconductor, Inc.
Fujitsu Computer Packaging Technologies, Inc.
Fujitsu Computer Products of America, Inc.
Fujitsu Microelectronics, Inc.
Fujitsu Network Switching of America, Inc.
Fujitsu Network Transmission Systems, Inc.
Fujitsu Networks Industry, Inc.
Fujitsu Systems Business of America, Inc.
Fujitsu Systems Business of Canada, Inc.
Open Systems Solutions, Inc.
Fujitsu Personal Systems, Inc.

EUROPE

Fujitsu Deutschland GmbH
Fujitsu España, S.A.
Fujitsu Europe Ltd.
Fujitsu Europe Telecom R&D Centre Ltd.
Fujitsu Finance (U.K.) PLC
Fujitsu France S.A.
Fujitsu International Finance (Netherlands) B.V.
Fujitsu Italia S.p.A.
Fujitsu Microelectronics Ireland Ltd.
Fujitsu Microelectronics Italia S.r.l.
Fujitsu Microelectronics Ltd.
Fujitsu Mikroelektronik GmbH
Fujitsu Nordic AB
Fujitsu Systems Business of Europe Ltd.
Fulcrum Communications Ltd.

LATIN AMERICA

Fujitsu do Brasil Ltda.
Fujitsu Vitória Computadores e Serviços Ltda.

FUJITSU LIMITED

6-1, Marunouchi 1-chome, Chiyoda-ku, Tokyo 100, Japan
Phone: National (03) 3216-3211 International (Int'l Prefix) 81-3-3216-3211 Telex: J22833 Cable: "FUJITSULIMITED TOKYO"
Facsimile: National (03) 3216-9352 International (Int'l Prefix) 81-3-3216-9352