PAPER *Special Section on Digital Signal Processing*

DSP Code Optimization Utilizing Memory Addressing Operation

Nobuhiko SUGINO[†], *Member*, Satoshi IIMURO[†], *Nonmember*, Akinori NISHIHARA[†], *and* Nobuo FUJII[†], *Members*

SUMMARY In this paper, DSPs, of which memory addresses are pointed by special purpose registers (address registers: ARs), are assumed, and metheds to derive an efficient memory access pattern for those DSPs are proposed. In such DSPs, programmers must take care for efficient allocation of memory space as well as effective use of registers, in order to derive an efficient program in the sense of execution period. In this paper, memory addresses and AR update operations are modeled by an access graph, and a novel memory allocation method is presented. This method removes cycles and forks in a given access graph, and decides an address location of variables in memory space with less overhead. In order to utileze multiple ARs, methods to assign variables into ARs are investigated. The proposed methods are applied to the compiler for DSP56000 and are proved to be effetive by generated codes for several examples.

key words: *DSP compiler, code optimization, memory addressing*

1. Introduction

A digital signal processor (DSP) is now used for various high-speed applications: mainly for real-time signal processing applications such as filters, FFTs, CODECs, MODEMs, etc.. Although DSPs are fast enough for realizing such applications, an efficient (short in execution time) program is required as well. Writing such a program is cost and time consuming work even for expert programmers. They are required to have both enough knowledge of the processor architecture and the processing algorithm, and, in order to write an efficient program, try as many programming techniques as they know, using primitive tools such as assembler or low level (machine level) languages. To reduce this heavy programming load, software tools such as high-level language and its compiler would be very helpful. Recently, DSP venders provide C compilers [1]–[3], and some researchers present programming tools or compilers for novel programming languages [4]. In one of these compilers, DIMPL (Digital network IMPlementation Language) and its compiler [5], [6] has been proposed. In these compilers, an efficient program benefits from effective use of registers. Efficient access of memory in a program, however, is as important as effective use of registers in order to derive an efficient program.

In many DSPs, memory content is accessed indirectly through address registers (ARs). Although AR must be updated before memory access, simple AR operations such as increment or decrement (±1) can be performed in one instruction cycle besides data operations such as addition, subtraction or data movement. When AR update amount is beyond increment or decrement, additional one instruction cycle is required for substituting memory address for AR (immediate AR load operation), which becomes overhead in a program. Therefore, in order to derive an efficient program, reduction in the number of such immediate AR loads is very important.

In [7], a method to share memory area with several variables is proposed. Although it can reduce total memory area and the AR update amount is expected to be small, the number of immediate AR loads sometimes increases. Assignment of memory addresses to variables is studied in [8],[9]. In this paper, both efficient memory address allocation and effective assignment of ARs are discussed, and algorithms to derive an efficient memory access pattern are proposed.

2. Memory Addressing

Whenever a datum in memory space is read or written as an operand, memory address corresponding to the datum must be pointed by some ways, which are called as memory addressing modes. In general purpose processors, several addressing modes such as immediate addressing mode, indirect addressing mode, indexed addressing mode, and so on, are provided so that programmers easily implement various data structures. These addressing modes are also utilized in compilers for general purpose processors. Compared with memory addressing modes for general purpose processors, those for DSPs are usually very much simplified in order to achieve high performance in computation, although some of DSPs have addressing modes for specific signal processing algorithms such as bit-reversed addressing in FFT.

Almost all the DSPs have indirect addressing modes: they have so called address registers (ARs) to point the memory addresses to be accessed. In this article, memory addressing of DSPs is assumed as below.

Manuscript received December 27, 1995.

Manuscript revised March 11, 1996.

[†] The authors are with the Department of Physical Electronics, Faculty of Engineering, Tokyo Institute of Technology, Tokyo, 152 Japan.

Table 1 Intermediate code for Fig. 1 and AR operations.

Code Cycle	Intermediate Code	Access	Address) AR operation			
			Conventional [7]	Efficient	Multiple ARs	
					AR0	AR1
1	$A \leftarrow N_0 \times m_0$	N_0	0) +1	0) *LD*		0) +1
2	$A \leftarrow A + N_1 \times m_2$	N_1	1) +1	2) +1		1) +1
3	$A \leftarrow A + N_2 \times m_1$	N_2	2) +1	3) *LD*	+1	2) +1
4	$N_3 \leftarrow A$	N_3	3	1		3
5	$A \leftarrow A \times m_3$		) *LD*	) +1		
6	$A \leftarrow A + N_1 \times m_5$	N_1	1) +1	2) +1	+1 (1	) +1
7	$A \leftarrow A + N_2 \times m_4$	N_2	2) *LD*	3) +1	(2	
8	$N_4 \leftarrow A$	N_4	4) *LD*	4) −1	(4	
9	$A \leftarrow N_2$	N_2	2) −1	3) −1	−1 (2	) −1
10	$N_1 \leftarrow A$	N_1	1) *LD*	2) −1	1	
11	$A \leftarrow N_3$	N_3	3) −1	1) *LD*		3) −1
12	$N_2 \leftarrow A$	N_2	2	3		2
Number of AR Loads			4	3	0	

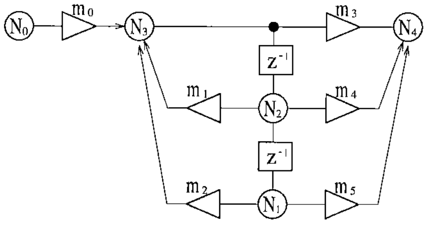


Fig. 1 SFG of 2nd order IIR filter.

Memory addressing of the DSP model

- Memory is accessed only by AR indirect addressing.
- AR can be increased or decreased by one (±1) besides arithmetic operation or data movement in one instruction cycle, only when memory is accessed in this cycle.
- Immediate AR load costs one instruction cycle by itself.

When an instruction requires to read a datum from memory or to write a datum into memory, AR must be updated before this instruction. If this update amount is beyond ±1, immediate load operation of the memory address into AR is required. This immediate AR load costs another instruction cycle, and becomes overhead of a program.

Let us think of memory address and AR operations with a simple example. The signal flow graph of a second order IIR filter is shown in Fig. 1. After an appropriate computational ordering, intermediate codes are derived as shown in left hand side of Table 1, where variables N_0–N_4 are kept in memory, and temporary variable A is kept in the register.

The right hand side in Table 1 shows the memory access sequences and AR operations. The conventional allocation method used in the DIMPL compiler decides the addresses of variables as they appear in the program. The decided addresses and AR operations for the example in Fig. 1 are shown at the column labeled as "Conventional" in Table 1. By this method, 4 immediate AR loads between codes 4–6, 7–8, 8–9, and 10–11, which are denoted as "*LD*s" in Table 1 are required. If memory address is given as the next column "Efficient," only 3 immediate AR loads between codes 1–2, 3–4, 11–12 are required, and one immediate AR load can be saved. Moreover, DSPs usually have two or more ARs, and utilization of these ARs can further improve memory addressing. For this example, after choosing an appropriate AR for every memory access no immediate AR load appears in the resultant program codes as shown in the rightmost column "Multiple ARs" in Table 1.

As shown in the above example, both a memory address allocation and an AR assignment are important. In this paper, the issue to find an efficient memory address allocation is called as "memory allocation problem" and, the issue to find an appropriate memory access assignment into multiple ARs is called as "AR assignment problem." Although programmers usually take care of these two problems at the same time and derive an efficient memory access pattern, memory allocation and AR assignment are considered as distinct problems in this paper in order to simplify the overall problem. Methods for memory allocation problem and AR assignment problem are mentioned in Sects. 3 and 4, respectively.

3. Memory Allocation

3.1 Access Graph (AG)

Memory access sequence consists of memory addresses and AR operations and is modeled by an access graph (AG) in this paper, where the memory addresses correspond to the variables in a program. For example, AG for Fig. 1 is shown in Fig. 2, where each node and edge denote the variable to be accessed and the required AR operations, respectively: all the nodes are labeled by corresponding variables "N_0"–"N_4," and the edge

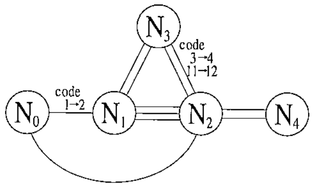
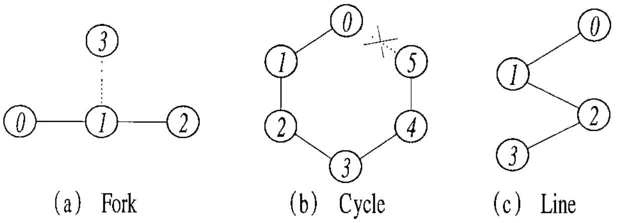


Fig. 2 The AG for Fig. 1.



(a) Fork (b) Cycle (c) Line

Fig. 3 The classes of AG.

corresponding to the AR operation from variable N_0 in code 1 to variable N_1 in code 2 is illustrated by the label "code 1→2." Although multiple edges appear between two nodes in AG, they are handled by one edge and its weight. In Fig. 2, 3 edges appear between variables N_2 and N_3, but they are considered as a edge with weight of 3.

3.2 Memory Allocation Algorithm

When a given AG is a line graph as shown in Fig. 3 (c), no immediate AR load is required and memory addresses for variables can easily be decided. However, in general cases, AGs include several fork nodes as shown in Fig. 3(a) or several cycles as shown in Fig. 3(b).

By removal of appropriate edges, cycles and forks in a given AG are eliminated and a given AG can be transformed into a line graph. In return, immediate AR loads are required at the memory access corresponding to the removed edges. Therefore the memory allocation problem is assumed as finding a line graph for a given AG with the minimum number of removed edges. To derive the optimal address allocation is an NP–hard problem, so that a heuristic method is employed to linearize a given AG.

In this heuristic method, the following parameters are taken into account in order to select the edges to be removed.

- Weights
  A weight of an edge $e = (u, v)$ (u,v: end nodes of e) is given by the number of AR operations required between the variable u and v.

$$w(e) = \begin{array}{l}\text{the number of required AR}\\ \text{operations at the edge } e\end{array} \tag{1}$$

- Forks
  An example of a fork is shown in Fig. 3(a). When k independent edges are connected to a node v, which is a fork node, at least $k-2$ of these edges must be removed, in order to derive a line graph as shown in Fig. 3(a). The fork count of a node v (fork(v)) is defined as:

$$\text{fork}(v) = \max\{\deg(v) - 2, \quad 0\}, \tag{2}$$

  where deg(v) denotes the number of independent edges connected to node v or "the order" of node v. For a fork of fork count k, at least k edges connected to fork node v must be removed.

- Cycles
  An example of a cycle is shown in Fig. 3(b). At least one edge in a cycle must be removed in order to derive a line graph as shown in Fig. 3(b). Therefore, cycle flag of edge e(cycle(e)) is defined as:

$$\text{cycle}(e) = \begin{cases} 1, & e \text{ is included in a cycle.} \\ 0, & e \text{ is not included} \\ & \quad \text{in any cycles.} \end{cases} \tag{3}$$

The cost function estimates how much a given AG is linearized after removal of $e = (u, v)$ (u,v: edge nodes of e), by evaluating above parameters. The cost function is given by

$$\text{benefit}(e) = \frac{\text{fork}(u) + \text{fork}(v) + \text{cycle}(e)}{w(e)}. \tag{4}$$

By the use of this cost function, algorithm to derive an efficient memory allocation, which is named as "ALOMA (Address LOad Minimization Algorithm)" follows.

Memory Allocation Algorithm: ALOMA

Input AG $G = (V, E)$
Memory area $M = \{x \mid x \in \boldsymbol{N};\ 0 \leqq x < |V|\}$

Output Memory allocation ($m : V \rightarrow M$) to minimize the cost function

Objective function The number of immediate AR load operations ($\text{load}_G(G)$) required in G

1. For all the nodes $v \in V$ in G, evaluate fork(v) by Eq. (2).
2. For all the edges $e \in E$, evaluate cycle(e) by Eq. (3).
3. For all the edges $e \in E$ in G, evaluate benefit(e) by Eq. (4)
4. Find a edge with the maximum benefit and remove it from G.
5. Repeat 1–4, until G becomes a line graph.
6. For the nodes in the derived line graph, memory addresses are decided.

Table 2 Comparison of immediate AR load operations.

Filters	Total Program Steps (without AR loads)	Total Number of Memory Accesses	Number of Variables	Conventional Methodin Ref. [7]	Proposed Method
3rd order lattice	48	27	11	15	14
5th order wave	111	56	25	25	18
11th order wave	321	145	64	89	67
17th order wave	510	232	103	143	110

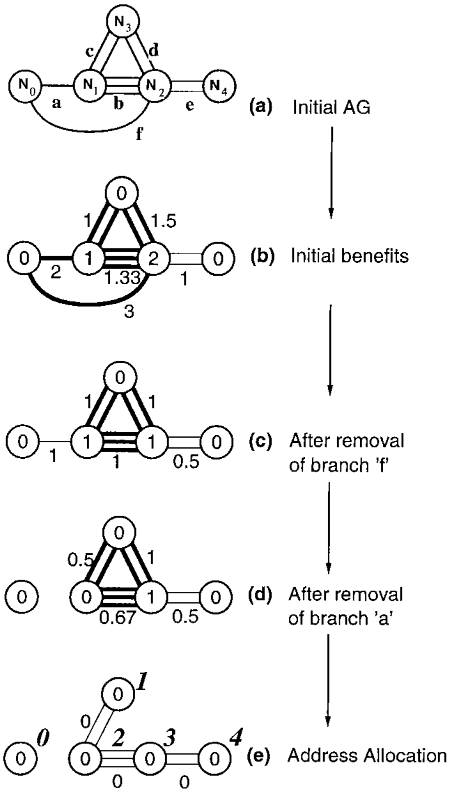


Fig. 4 Procedure of memory allocation.

The procedure of the proposed method is explained by using the example shown in Fig. 2. Whole the procedure is shown in Fig. 4, where the numbers labeled on nodes and edges denote fork counts and benefits respectively, and edges with bold lines indicate cycles.

Figure 4 (a) shows the initial AG. At first, edge "f" is removed from the initial AG (Fig. 4 (b)), because this edge has the maximum benefit "3" in all edges. After this removal, fork counts of nodes and benefits of edges are renewed for the new AG (Fig. 4 (c)). In this AG, edges "a," "b," "c" and "d" have the same benefits "1," and edge "a" is assumed to be removed(Fig. 4(d)). Then, edge "d" of the maximum benefit "1" in AG (Fig. 4(d)) is removed. Finally, benefits of all the remaining edges become "0," and a line graph Fig. 4(e) appears. For this line graph, memory addresses of node N_0 through node N_3, which are labeled as superscripts on nodes, are decided.

3.3 Code Generation Result

The proposed method is applied to the DIMPL compiler [5], [6]. This compiler reads the filter network description, decides the computational ordering (scheduling) and then generates an efficient code. The code generation part is closely related to the target DSPs, while the rest parts are independent of processors. Slight modification of the code generation part can give the compiler for different DSPs. Compilers are now available for μPD7720/25, μPD77230 (NEC), TMS32010/15/20/25 (TI) and DSP56000 (Motorola). DSP56000 (Motorola) [10] is chosen for the target DSP, because it has only 2 arithmetic registers and memory access is frequently required in its programs. The proposed method is implemented in the code generation part of the compiler.

For several filter examples, DSP codes are generated by this compiler, and are compared with the conventional DIMPL compiler. These filters include rather complicated memory access sequences, and they are appropriate for the comparison in memory addressing methods.

Table 2 shows the comparison of the proposed method to the conventional method in terms of the number of immediate AR load operations included in the generated codes. In the table, total program steps, the total number of memory accesses and the number of variables for example filters are also shown. From the table, the number of immediate AR load operations is reduced up to 20%.

4. AR Assignment

4.1 AR Assignment Algorithms

As mentioned above, AR assignment and memory allocation is performed separately in the case of DSPs with multiple ARs. Furthermore, for the simplicity, every variable is assigned to one of ARs, rather than every access is assigned to an AR, so that ALOMA algorithm is applied after the AR assignment. This limits memory space accessed by each AR, and reduces memory access pattern choices. Memory access patterns derived under this limitation, however, are efficient enough for many examples.

Suppose a given program has a set V of variables on memory space, and they are accessed by k ARs of

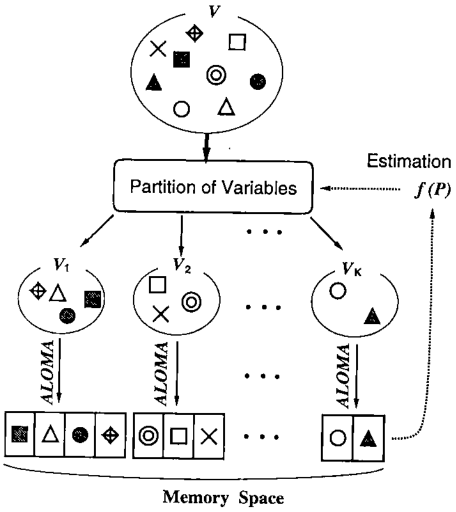


Fig. 5 AR assignment diagram.

a target DSP. AR assignment algorithm partitions the variables V into k subsets of variables (V_1 through V_k), so that each group is accessed by an individual AR. For every subset, the memory allocation is decided by the ALOMA algorithm.

The goal of AR assignment is to find a partition $P = \{V_1, V_2, \ldots, V_k\}$ with the least immediate AR loads. By examining all the possible partition P, optimal AR assignment with the least immediate AR loads can be found, but its computation costs too much. Therefore, practically, a heuristic method is employed to derive the partition of variables. The diagram of AR assignment algorithm is shown in Fig. 5. Partition and memory allocation are iteratively performed in the algorithm, where $f(P)$ represents the quality of partition P, and controls partitioning at the next iteration. In this paper, $f(P)$ is given by the number of immediate AR loads at first.

4.2 AR Assignment Based on Min-Cut Algorithm

AR assignment problem is partitioning variables into subsets so as to mininimize cost $f(P)$ or the total number of AR loads. It is quite similar to well-known min-cut problem, which is dividing a graph into two subgraphs so as to minimize the sum of the costs on all edges. The heuristic iterative method for solving min-cut problem proposed by Fiducca and Mattheyses [12], [13] is applied to AR assignment problem.

Min-cut algorithm, which partitions a set of the variables V into two subsets, A and B, is shown below. In this algorithm, gain() is employed to show how much partitioned subsets are improved after the movement of a variable. When variable $a \in A$ moves to B, gain(a) is defined by,

$$\text{gain}(a) = f(\{A, B\}) - f(\{A - \{a\},\ B + \{a\}\}). \quad (5)$$

In the following min-cut algorithm, evaluation of this function and movement of variable are iteratively performed and finally one of the best partitioning is derived.

Min-cut algorithm

1. Decide the initial partition $P_0 = \{A_0, B_0\}$. (iteration cycle $i = 0$).
2. Evaluate the gain() for all the variables in P_i.
3. Select the variable with the maximum gain ($= g_{i+1}$) among not-yet-moved variables in P_i.
4. Move the selected variable from the current subset to the other subset.
5. If all the variables have been selected and moved, go to step 6. Otherwise, let $i \leftarrow i + 1$ and return to step 2.
6. Choose k to maximize $\sum_{i=1}^{k} g_i$, and let $g_{\max} = \sum_{i=1}^{k} g_i$.
7. If $g_{\max} > 0$, partition P_k is better than the initial partition P_0. Let P_k be a new initial partition P_0 and go back to step 2. Otherwise, $g_{\max} = 0$ and the optimal partition is derived in P_0.

Since the min-cut algorithm divides the given set of variable into two subsets, partition into 3 or more subsets requires iterative use of this algorithm. The algorithm to partition variables into k subsets is shown below.

***k*-partition algorithm**

1. Select a pair of subsets V_i, V_j among all the subsets $\{V_1, V_2, \ldots, V_K\}$.
2. Apply min-cut algorithm to this pair.
3. Repeat step 1–2 until no movement of variable between $V_1, V_2, \ldots, V_K$ appears.

Selection of a pair seems very important in this algorithm. Although several strategies are tested, the number of AR loads in the resultant code is not much improved. Rather than selection of a pair, strategy for initial partition has an effect on the resultant code. The following two strategies are examined.

1. Assign all the variables into one of the subsets (all other subsets are empty).
2. Assign variables into subsets randomly.

The resultant codes shows the first strategy is about 20% superior to the second strategy, so that the first strategy is chosen in this paper.

Table 3 Result of AR assignment based on min-cut algorithm (number of immediate AR loads).

Filters	Number of ARs			
	1	2	3	4
3rd order lattice	14	3	1	0
5th order wave	18	7	4	3
11th order wave	67	28	17	13
17th order wave	110	44	39	20

Table 4 Result of AR assignment based on SA. (number of immediate AR loads).

Filter	Number of ARs			
	1	2	3	4
3rd order lattice	14	3	0	0
		3.0	0.0	0.0
		3	0	0
5th order wave	18	7	4	1
		7.0	4.0	1.1
		8	5	2
11th order wave	67	25	15	10
		26.1	16.6	11.5
		27	18	15
17th order wave	110	40	25	16
		43.6	29.4	21.3
		47	34	26

4.3 Code Generation Result

The above mentioned method is applied to DIMPL compiler for DSP56000, which is mentioned in the former chapter, and the numbers of immediate AR loads in generated codes are tabulated in Table 3.

For the comparison, a partition method based on simulated annealing technique (SA) is applied, and the code generation results are shown in in Table 4, where the best, the average and the worst results in 500 trials are tabulated in first, second and third row, respectively for each case.

From these tables, the AR assignment based on min-cut algorithm is comparable to that based on SA in terms of the number of immediate AR loads. The AR assignment based on SA however takes longer time than that based on min-cut algorithm. Moreover, by the AR assignment based on SA, the number of immediate AR loads changes for each trial. Consequently, the AR assignment method based on min-cut algorithm is said to be better.

4.4 An Improved Cost Function

In the above mentioned methods, the number of immediate AR loads is employed for cost function $f(P)$, and it is estimated by ALOMA algorithm, which costs $O(n^3)$ of computational time (n: the number of variables). Therefore, a simple method to estimate the number of AR loads is desired.

The rank of cycle in a graph G gives the minimum number of edges to be removed from a graph, when the graph is linearized. It is estimated as

$$\nu(G) = |E| - |V| + \omega(G), \tag{6}$$

where $|E|$, $|V|$ and $\omega(G)$ represent the numbers of edges, nodes and connections in G, respectively. When $|V|$ and $\omega(G)$ in a given AG G are assumed to be constant, the number of edges to be removed is related only to $|E|$, and the novel cost function is given by

$$f(P) = \sum_i |E(G_i)|, \tag{7}$$

where $|E(G_i)|$ represents the number of edges in AG G_i, and G_i denotes the access graph formed by the partitioned subset V_i. By use of this novel cost function, estimating the number of AR loads costs $O(n)$ of computational time. In return, this cost function does not always give the precious estimation of AR loads, especially when a given AG includes no cycle but includes forks. Such an AG, however, rarely appears in many examples.

This cost function $f(P)$ is implemented in AR assignment part of compilers, which work on NWS-5000 SB (Sony). Table 5 shows the comparison of two cost functions in terms of the computational times for several filter examples. From the table, AR assignment by this novel cost function is performed in remarkably short time. The number of AR loads in a derived code, however, increases slightly.

Table 5 Computation time for partitioning (second) into 4 subsets by method based on min-cut.

filters	cost based on ALOMA	A novel cost
11th order wave	427	0.7
17th order wave	6,747	3.0

5. Conclusion

In this paper, methods to derive an efficient memory access pattern for DSPs, of which memory space is indexed by address registers (ARs), are proposed. First, memory location and access operations are discussed. They are modeled by an access graph, and, by removing cycles and fork nodes from this graph, an efficient memory allocation is derived. This method is applied to DIMPL compiler for DSP56000 and generated code can save up to 20% of immediate AR loads.

Second, DSP with multiple address registers are assumed and an efficient memory access pattern for these DSPs can be derived by use of the methods to assign variables into several ARs together with the proposed memory allocation method. These methods are applied to the DIMPL compiler, and from the code generation results, the AR assignment method based on min-cut algorithm is found to be superior to that based on simulated annealing techniques. Improvement in computational period for AR assignment is also studied in this article. Methods to derive further efficient memory access pattern must be investigated.

Acknowledgement

The authors are grateful to Ass. Prof. S. Takagi of Tokyo Institute of Technology, for the valuable discussions. This work is a part of the Research Body of CAD21 (Computer Aided Design for 21th Century) in Tokyo Institute of Technology.

References

[1] R. Simar Jr. and A. Davis, "The application of hign-level langueges to single chip digital signal processors," 1988 ICASSP, pp.1678–1681, 1988.

[2] Texas Instruments, "TMS320C30 C Compiler—Reference Guide," 1990.

[3] J. Hartung, S.L. Gay, and S.G. Haigh, "A practical C language compiler/optimizer for real-time implementations on a family of floating Point DSPs," 1988 ICASSP, pp.1674–1677, 1988.

[4] E.A. Lee, Wai-Hung Ho, et al., "Gabriel: A design environment for DSP," IEEE Trans. ASSP, vol.37, no.11, pp.1751–1761, Nov. 1989.

[5] N. Sugino, A. Toshikiyo, E. Watanabe, and A. Nishihara, "Computational ordering of digital signal processing networks and its application to compilers for signal processors," IEICE Trans., pp.327–335, 1988.

[6] N. Sugino, S. Ohbi, and A. Nishihara, "Computational ordering of digital network under the pipeline constraints and its application to compiler for DSPs," Proc. ECCTD '89, pp.395–399, Sept. 1989.

[7] S. Ohbi, N. Sugino, and A. Nishihara, "Automatic DSP code generation with effective utilization of storage resources—Improvement in DSP compiler DIMPL—," Proc. of 4th Digital Signal Processing Symposium, Dec. 1989.

[8] S. Iimuro, N. Sugino, A. Nishihara, and N. Fujii, "Code optimization method utilizing memory addressing and its application to DSP compilers," Proc. IEICE Fall Conf. A–112, 1992.

[9] S. Iimuro, N. Sugino A. Nishihara, and N. Fujii, "Code optimization method utilizing memory addressing operation and its application to DSP compiler," Proc. of Asia-Pacific Conference on Circuit and Systems, pp.151–156, 1994.

[10] MOTOROLA, "DSP56000/DSP56001 Digital Signal Processor User's Manual," 1990.

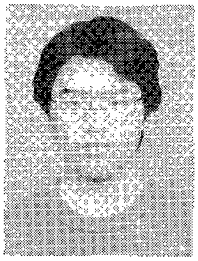
[11] NEC, "μPD77230 Users Manual," 1987.

[12] B.W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," Bell System Technical Journal, vol.49, pp.291–307, Feb. 1970.

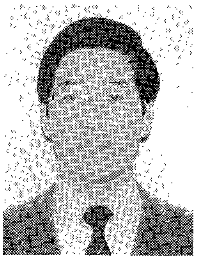
[13] C.M. Fiducca and R.M. Mattheyses, "A linear-time heuristic for improving network partitions," Proc. 19th DAC, pp.175–181.

[14] S. Kirkpatrick, "Optimization by simulated annealing," Science, vol.220, no.4598, pp.671–680, 1983.

Nobuhiko Sugino was born in Yokkaichi, Mie, Japan on November 19, 1964. He received B.E., M.E. and Dr. Eng. degrees in physical electronics from Tokyo Institute of Technology in 1987, 1989 and 1992, respectively. Since 1992, he has been with Tokyo Institute of Technology, where he is now a Research Associate of Department of Physical Electronics, Faculty of Engineering. His main research interests are in hardware and software for digital signal processing, especially in software development tools for digital signal processors. Dr. Sugino is a member of IEEE.



Satoshi Iimuro was born in Yokohama, Japan on May 23, 1968. He received B.E. and M.E. Eng. degrees in physical electronics from Tokyo Institute of Technology in 1992 and 1994, respectively. Since 1994, he has been with Hitachi, Ltd., where he is now a Researcher of Multimedia System Research and Development Department. His main research interests are in development of digital televison receiver, especially in hardware and software of receiver for Satellite digital broadcasting.



Akinori Nishihara was born in Fukuoka, Japan, on February 26, 1951. He received the B.E., M.E. and Dr. Eng. degrees in electronics from Tokyo Institute of Technology in 1973, 1975 and 1978, respectively. Since 1978 he has been with Tokyo Institute of Technology, where he is now Associate Professor of Department of Physical Electronics, Faculty of Engineering. His main research interests are in filter design, 1D and multi-D signal processing, and modern applications of classical circuit theory. From 1990 to 1994 he served as an Associate Editor of the IEICE Trans. Fundamentals, and is now serving as an Associate Editor of the IEEE Trans. Circuits & Systems II. He is now Student Activities Committee Chair, IEEE Region 10 (Asia Pacific Region). Dr. Nishihara is a member of IEEE, EURASIP, ECS and JET.

Nobuo Fujii received B.E. degree from Keio University, Yokohama, Japan, and M.E. and Doctor of Engineering degrees from Tokyo Institute of Technology, Tokyo, Japan, in 1966, 1968, and 1971, respectively. Since 1971, he has been with the Faculty of Engineering, Tokyo Institute of Technology where he is now a professor in the Department of Physical Electronics. From 1984 to 1985, he was a visiting scholar at the University of California, Santa Barbara. From 1990 to 1992, he served as an editor of the Transaction of the Institute of Electronics, Information, and Communication Engineers and is now one of the chief editors of the International Journal of Analog Integrated Circuits and Signal Processing, Kluwer Academic Publishers. He is the chairman of the technical group of electronic circuits of IEE Japan and the chairman of the Circuits and Systems Society of IEEE Tokyo Chapter. His main interest lies in the fields of active networks, analog integrated circuits, and analog signal processing. He is the recipient of the Best Paper Award of the Institute of Electrical and Communication Engineers of Japan. He is the author of more than 10 books. Dr. Fujii is a member of the Institute of Electrical and Electronics Engineers and the Institute of Electrical Engineers of Japan.