PAPER Special Section on Multiple-Valued Logic and VLSI Computing

A Packet Classifier Based on Prefetching EVMDD (*k*) Machines

Hiroki NAKAHARA[†a)], **Tsutomu SASAO**[††b)], ***and*** **Munehiro MATSUURA**[†††c)], ***Members***

SUMMARY A Decision Diagram Machine (DDM) is a special-purpose processor that has special instructions to evaluate a decision diagram. Since the DDM uses only a limited number of instructions, it is faster than the general-purpose Micro Processor Unit (MPU). Also, the architecture for the DDM is much simpler than that for an MPU. This paper presents a packet classifier using a parallel EVMDD (*k*) machine. To reduce computation time and code size, first, a set of rules for a packet classifier is partitioned into groups. Then, the parallel EVMDD (*k*) machine evaluates them. To further speed-up for the standard EVMDD (*k*) machine, we propose the prefetching EVMDD (*k*) machine which reads both the index and the jump address at the same time. The prefetching EVMDD (*k*) machine is 2.4 times faster than the standard one using the same memory size. We implemented a parallel prefetching EVMDD (*k*) machine consisting of 30 machines on an FPGA, and compared it with the Intel's Core i5 microprocessor running at 1.7GHz. Our parallel machine is 15.1–77.5 times faster than the Core i5, and it requires only 8.1–58.5 percents of the memory for the Core i5.

key words: *many core, packet classification, decision diagram, multi-valued logic*

1. Introduction

1.1 Packet Classification

A packet classification [22] is a key technology in routers and firewalls. A packet header includes a protocol number, a source address, a destination address, and a port number [6]. The packet classifier performs a predefined action for a corresponding rule. Applications for the packet classifier include a firewall (FW), an access control list (ACL), and an IP chain for an IP masquerading technique. The throughput for the state-of-the-art packet classifier using an MPU is at most hundreds mega bits per second [4], so it cannot keep up with accelerated speed up of the Internet.

This paper proposes the parallel edge-valued multi-valued decision diagram (EVMDD) machine that is a kind of the decision diagram machine [1], [2], [24]. The decomposed packet classification tables are represented by multiple EVMDDs, and it evaluates them in a parallel. The EVMDD machine uses a single instruction to evaluate the EVMDD. Thus, its architecture is simpler than that for the MPU, and it has the dedicated branch instructions that are extensively used in the packet classifier [18]. Thus, the parallel EVMDD machine is faster than the MPU.

Manuscript received December 6, 2013.
Manuscript revised April 8, 2014.

[†]The author is with the Faculty of Engineering, Kagoshima University, Kagoshima-shi, 899–0065 Japan.
[††]The author is with the Department of Computer Science, Meiji University, Kawasaki-shi, 214–8571 Japan.
[†††]The author is with the Department of Computer Science and Electronics, Kyushu Institute of Technology, Iizuka-shi, 820–8502 Japan.
a) E-mail: nakahara@eee.kagoshima-u.ac.jp
b) E-mail: sasao@cse.kyutech.ac.jp
c) E-mail: matsuura@cse.kyutech.ac.jp
DOI: 10.1587/transinf.2013LOP0020

1.2 Contributions of the Paper

This paper is the update version of the previous publication [13]. Contributions of the previous version were as follows:

1 *Proposed the parallel EVMDD machine for the packet classifier:* A packet classification circuit based on an EVMDD has been proposed [14]. Different users require systems with different performance. Thus, different architecture should be used. For low-end users including SOHO (small office and home office), the embedded processors or the general purpose processors are used. So, this paper proposed a special purpose processor based on an EVMDD. Its parallel architecture and the dedicated instruction is suitable for the packet classification.
2 *Obtained the parameters for an optimal EVMDD machine:* Since the EVMDD evaluates 2^k-valued variables, several size k of the EVMDD exists. This paper obtained parameters for the an optimal EVMDD machine with respect to the memory size and the delay time.
3 *Compared with Intel's Core i5 processor.*

Additionally, this paper contributes as follows:

1 *Applied the prefetching method to reduce a delay time:* In this paper, we applied the prefetching method to the EVMDD (*k*) machine, which is smaller and faster than the MTMDD (*k*) machine [13].
2 *Analyze the delay time of the parallel prefetching EVMDD (k) machine:* The previous work only showed that the parallel EVMDD (*k*) machine is faster than the Intel's Core i5 processor by means of experiments. This paper analyzed the delay time for both the parallel EVMDD (*k*) machine and the Intel'l Core i5 processor.

1.3 Organization of the Paper

The rest of the paper is organized as follows: Section 2 defines the packet classifier; Section 3 introduces the standard

EVMDD (k) machine; Section 4 proposes the prefetching EVMDD (k) machine; Section 5 shows the realization of the packet classifier using the parallel prefetching EVMDD (k) machine; Section 6 compares the implemented machine with the Intel's Core i5; and Sect. 7 concludes the paper.

2. Packet Classifier

2.1 5-tuple Packet Classification

A packet classification table consists of a set of **rules**. Each rule has five input **fields**: Source address (SA), destination address (DA), source port (SP), destination port (DP), and protocol number (PRT). It also generates a **rule number** (Rule). A field has **entries**. In this paper, since we consider a realization of the packet classifier for the Internet Protocol version 4 (IPv4), we assume that SA and DA have 32 bits, DP and SP have 16 bits, and PRT has 8 bits. An entry for SA or DA is specified by an IP address; that for SP or DP is specified by **a closed interval [x,y]**, where x and y denote a port number such that $x \leq y$; and that for PRT is specified by a protocol number. SA and DA are detected by **a longest prefix match**; SP and DP are detected by **a range match**; and PRT is detected by **an exact match**. **A packet classifier** detects matched rules using the packet classification table. When two or more rules are matched, it selects a rule having the highest **priority**. In this paper, we assume that the rule with a larger number has a higher priority. Note that, any packet matches **a default rule** whose rule number is zero. Obviously, the default rule has the lowest priority.

Example 2.1: Table 1 shows an example of a packet classification table, where an asterisk '*' in an entry matches both 0 and 1, while a dash '-' in a field matches any pattern. In Table 1, each field has only four bits, smaller than the actual number of bits to simplify the example.

Consider the packet classification table shown in Table 1. The packet header with $SA = 0000$, $DA = 1010$, $SP = 8$, $DP = 8$, and $PRT = TCP$ matches rule 3, rule 1, and the default rule. Since the rule 3 has the highest priority, the output is 3. ■

2.2 Decomposition of a Packet Classification Table

Let p be the number of rules. Since $|X_{SA}| = |X_{DA}| = 32$, $|X_{SP}| = |X_{DP}| = 16$, and $|X_{PRT}| = 8$, the direct memory realization requires $2^{104}\lceil log_2(p+1)\rceil$ bits, which is too large to implement by a single memory. We decompose the packet classification table into field functions and a rule function (**Cartesian product method** [21])†.

An entry of a rule can be represented by **an interval function** [20]:

$$IN(R : A, B) = \begin{cases} 1 & (A \leq R \leq B) \\ 0 & (otherwise) \end{cases}$$

†In [21], Cartesian product was called "cross product".

Table 1 An example of a packet classification table.

in					out
SA	DA	SP	DP	PRT	Rule (Action)
1000	110*	[1,8]	[8,9]	ICMP	4
00**	1***	[2,9]	[6,8]	TCP	3
010*	0010	[8,15]	[7,14]	UDP	2
0***	10**	[8,9]	[4,11]	TCP	1
****	****	[0,15]	[0,15]	-	0 (default)

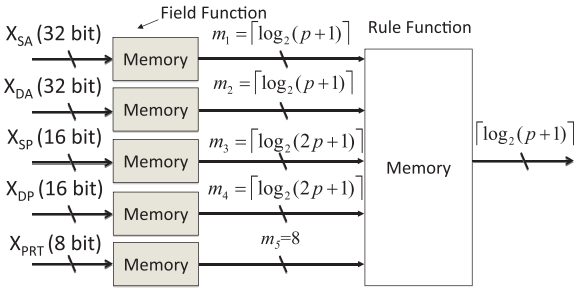

Fig. 1 Realization of decomposed packet classification table by memories.

where R, A, and B are integers. Let $x_i \in \{0, 1\}$, $y_i = *$, $v = (x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_m)$, and $A = \sum_{i=1}^{n} x_i 2^{i-1}$. Any entry for SA is represented by $IN(R_{SA} : A2^m, (A+1)2^m - 1)$. Similarly, each entry for DA can be represented by an interval function. Let b be the protocol number. Each entry for PRT is represented by $IN(R_{PRT} : b, b)$.

As shown in Example 2.1, multiple rules may match in a packet classification table. To resolve such a case, we use **a vectorized interval function**. Let r be the number of rules. A vectorized interval function is $\vec{H}(R) = \bigvee_{i=1}^{r} \vec{e}_i IN(R : A_i, B_i)$, where $\vec{e}_i$ is a unit vector with r elements, and only the i-th bit is one and the other bits are zeros.

For each value of $\vec{H}(R)$, we assign **a segment**, which is an interval or a set of intervals. Then, we define **a field function** $F(R) = I_i$, which generates an unique integer index I_i corresponding to the i-th segment $[C_i, D_i]$ satisfying $C_i \leq R \leq D_i$. Note that, to distinguish from an interval, we denote a segment consisting of an interval $[C, D]$ as $[C : D]$. Next, we define **a rule function** $G : Y \rightarrow R$, where $Y = I_1 \times I_2 \times \cdots \times I_k$ is the Cartesian product of sets of indices generated by field functions. As shown in Fig. 1, the packet classification table is decomposed into field functions and a rule function, and they are realized by memories.

In general, we can assign an arbitrary index to a segment. In this paper, we assign indices to make an **M_1-monotone increasing function** [11] to reduce the amount of memory. Let I be a set of integers including 0. An integer function $f(X) : I \rightarrow R$ such that $0 \leq f(X+1) - f(X) \leq 1$ and $f(0) = 0$ is **an M_1-monotone increasing function** on I. That is, for an M_1-monotone increasing function $f(X)$, $f(0) = 0$, and the increment of X by one increases the value of $f(X)$ at most by one.

Example 2.2: Figure 2 shows an example of segments for SA and DP shown in Table 1. Note that, rules are represented by intervals. ■

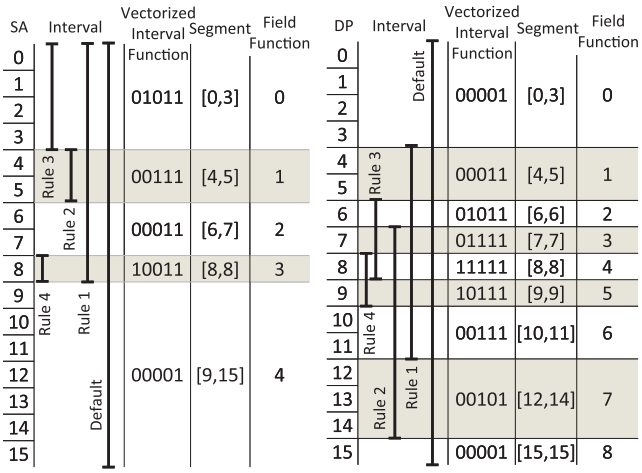

Fig. 2 An example of segment.

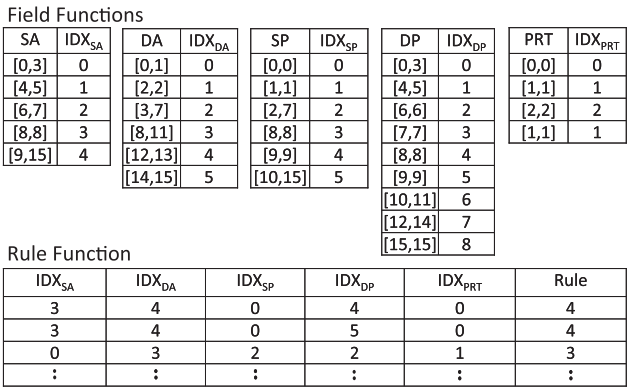

Fig. 3 Decomposition of a packet classification table into field functions and the rule function.

Example 2.3: Figure 3 shows the decomposition of the packet classification table shown in Table 1. As for the PRT field, we assigned "0" to ICMP, "1" to TCP, and "2" to UDP. Note that, we only show a part of the rule function due to the space limitations. ■

3. Standard EVMDD (k) Machine

3.1 MTMDD (k)

In this part, we introduce the EVMDD (k) machine that is a variant of the MTMDD (k) machine. First, we introduce the MTMDD (k).

Definition 3.1: A **Binary Decision Diagram (BDD)** [3], [10] is obtained by repeatedly applying of the **Shannon expansions** to a logic function f. Each non-terminal node labeled with a variable x_i has two outgoing edges which indicate nodes representing the cofactors of f with respect to x_i. **A Multi-Terminal BDD (MTBDD)** [5] is an extension of a BDD and represents an integer-valued function. In the MTBDD, the terminal nodes are labeled by integers.

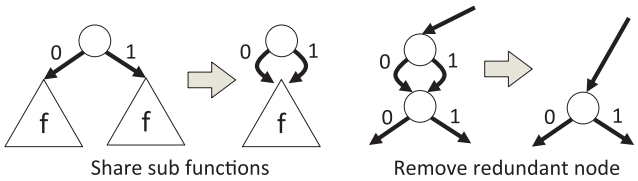

Fig. 4 Reduction rules for an MTBDD.

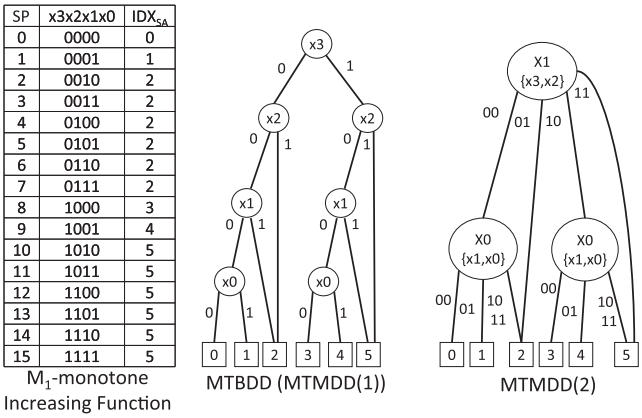

Fig. 5 Representation of an integer function by an MTMDD (k).

Definition 3.2: Let $X = (X_1, X_2, \ldots, X_u)$ be a partition of the input variables, and $|X_i|$ be the number of inputs for X_i. X_i is called **a super variable**. When the Shannon expansions are performed with respect to super variables X_i, where $|X_i| = k$, all the non-terminal nodes have 2^k edges. In this case, we have a **Multi-Terminal Multi-valued Decision Diagram (MTMDD(k))** [7]. Note that, an MTMDD(1) means an MTBDD. **The width of the MDD (k) at the height** k is the number of edges crossing the section of the MDD (k) between super variables X_{i+1} and X_i, where the edges incident to the same node are counted as one.

Example 3.4: Figure 5 shows an MTMDD (2) and an MTBDD representing the field function for SP shown in Fig. 3. When the input is $(x_3, x_2, x_1, x_0) = (1, 0, 1, 0)$, the output is 5. ■

3.2 Standard EVMDD (k) Machine

In this paper, we propose the prefetching EVMDD (k) machine, which is an improved version of the standard EVMDD (k) machine [13]. In this part, first we introduce the EVMDD (k). Then, we introduce the standard EVMDD (k) machine.

Definition 3.3: An Edge-Valued Binary Decision Diagram (EVBDD) [8], [9] is a variant of an MTBDD. An EVBDD consists of one terminal node representing zero and non-terminal nodes with weighted 1-edges, where the weight has an integer value α. An EVBDD is obtained by recursively applying the conversion shown in Fig. 6 to each non-terminal node in an MTBDD. Note that, in the EVBDD, 0-edges have zero weights.

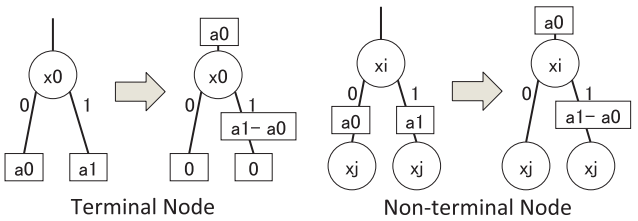

Fig. 6 Reduction rule for an EVBDD.

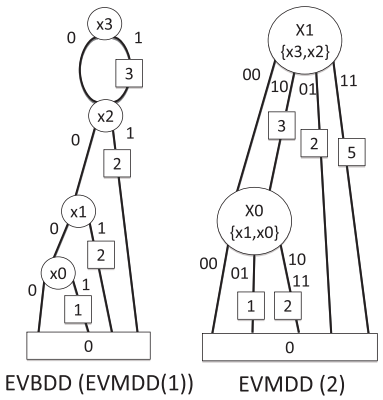

Fig. 7 An example of EVMDD (2).

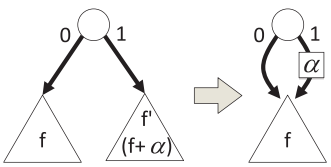

Fig. 8 Principle of reduction of nodes in an EVBDD.

Definition 3.4: An Edge-Valued MDD (k) (EVMDD (k)) [12] is a variant of the MDD (k), and represents a multi-valued input integer valued function. It consists of one terminal node representing zero and non-terminal nodes with edges having integer weights, and 0-edges always have zero weights.

Example 3.5: By applying the conversion rule shown in Fig. 6 to the MTBDD shown in Fig. 5, we get the EVMDD (2) shown in Fig. 7. Assume that the input is $(x_3, x_2, x_1, x_0) = (1, 0, 1, 0)$. When $X_1 = (1, 0)$, its edge weight is 3. Next, when $X_0 = (1, 0)$, its edge weight is 2. Thus, the sum of edge weights is 5. It is equal to the output of the MTMDD (2) in Example 3.4. ■

Suppose that a subfunction f' is obtained by adding α to a subfunction f. In an EVBDD, the number of nodes can be reduced by sharing f and f' with α edge (Fig. 8). The MTBDD can share only prefixes, while the EVBDD can share both prefixes and postfixes. A packet classification table can be represented by an M_1-monotone increasing function. Thus, the packet classification table can be efficiently represented by an EVBDD [14]. Since an EVMDD (k) has a weight in an edge, we use **a standard branch&output instruction** shown in Fig. 9.

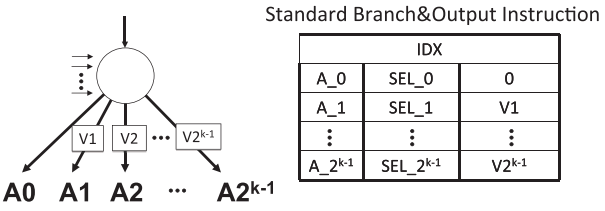

Fig. 9 Standard branch&output instruction to evaluate EVMDD (k).

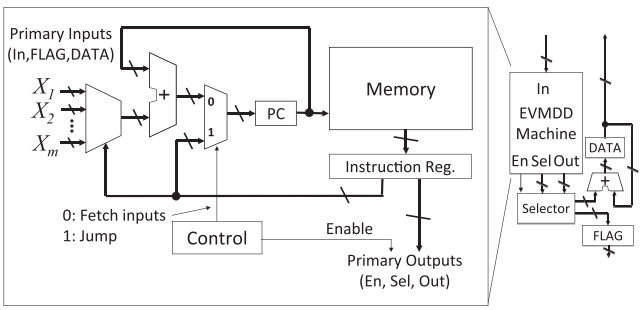

Fig. 10 EVMDD (k) machine.

A standard EVMDD (k) machine uses a standard branch&output instruction only. Figure 10 shows the standard EVMDD (k) machine. In Fig. 10, **The instruction memory** stores the instructions for an EVMDD (k); **the Instruction Register** stores the instruction from the instruction memory; **the Program Counter (PC)** retains the address for the instruction memory. The control circuit controls the branch&output operation.

As shown in Fig. 9, the standard EVMDD (k) machine has following two registers: **The data register (DATA)** retains the output, and **the flag register (FLAG)** retains the state to synchronize EVMDD (k) machines. Following the value of **the select field (SEL)** consisting of two bits, values of the DATA and SEL are updated.

SEL=00: $DATA \leftarrow 0$ (Clear DATA).
SEL=01: $DATA \leftarrow Output$.
SEL=10: $FLAG \leftarrow 0$ (Clear FLAG).
SEL=11: $FLAG \leftarrow Output$.

The execution of the indirect branch instruction and the standard branch&output instruction are performed by the following:

Algorithm 3.1: (branch&output instruction for the standard EVMDD (k) machine)

1. Execute the fetch mode.
   - 1.1 Read the index corresponding to the *IDX* field (Fig. 11 (a)).
   - 1.2 Add it to the PC (Fig. 11 (b)).
2. Execute the jump&output mode.
   - 2.1 Read the jump address corresponding to the PC.
   - 2.2 Set the jump address to the PC, and output the output value (Fig. 11 (c)).

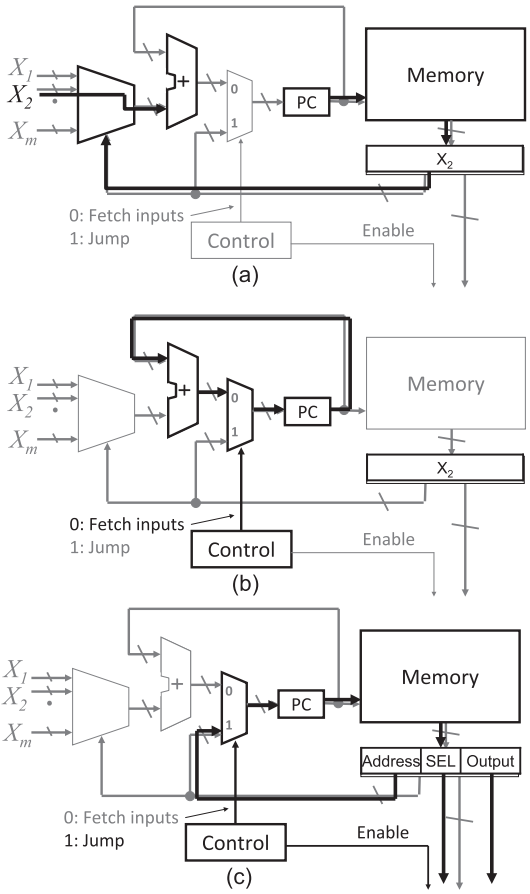

Fig. 11 Operation for the standard EVMDD (*k*) machine.

3. Terminate.

4. Prefetching EVMDD (*k*) Machine

In the standard EVMDD (*k*) machine shown in Fig. 10, to perform a branch operation, first, it reads the corresponding index. Then, it reads the jump address. Thus, the standard EVMDD (*k*) machine accesses the instruction memory twice. In the reduced ordered decision diagram, the jump address for the node and its corresponding index can be read at the same time [16]. Figure 12 shows **the prefetching branch&output instruction** which stores the jump address and its index in the same word. In this part, we propose **a prefetching EVMDD (*k*) machine** that needs to access the instruction memory only once. Thus, the prefetching EVMDD (*k*) machine is faster than the standard EVMDD (*k*) machine. The disadvantages for the prefetching EVMDD (*k*) machine are longer instruction words and increase of the memory.

Figure 13 shows the prefetching EVMDD (*k*) machine. In Fig. 13, the instruction memory, the instruction register, the output register, and the PC are the same as the standard EVMDD (*k*) machine shown in Fig. 10. The prefetching EVMDD (*k*) machine uses **the Prefetching Register** to store the prefetched index. As shown in Fig. 12, the

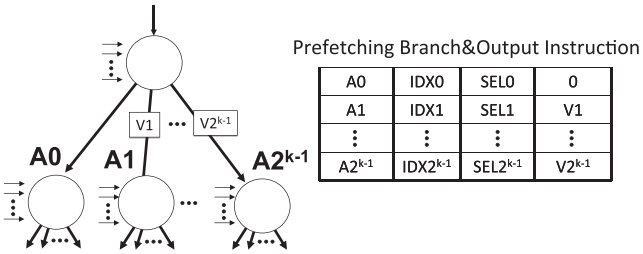

Fig. 12 Prefetching branch&output instruction.

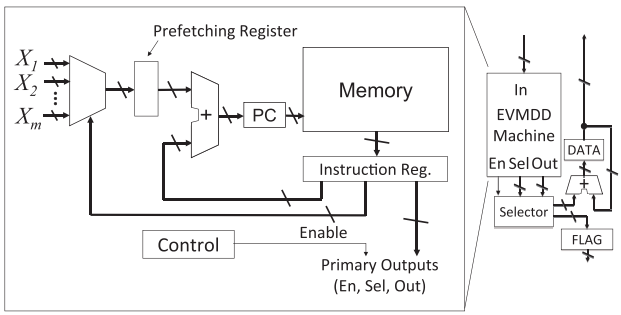

Fig. 13 Prefetching EVMDD (*k*) machine.

prefetching EVMDD (*k*) machine reads the jump address, the index, and the edge-value at the same time. It performs the jump, the index fetch, and the output operations in parallel. The following algorithm shows the operation of the prefetching EVMDD (*k*) machine.

Algorithm 4.2: Figure 14 shows the execution of the prefetching branch&output instruction of the prefetching EVMDD (*k*) machine.

1. Read the instruction memory specified by the PC (Fig. 14 (a)).
2. Perform following operations in parallel.
   - 2.1 The *PC* stores the indirect jump address which is obtained by summing the prefetching index and the jump address (*Adr*) (Fig. 14 (b)).
   - 2.2 The prefetching register stores the jump address (Fig. 14 (c)).
   - 2.3 Read the output (*V*), then store it to the output register specified by *SEL* (Fig. 14 (d)).

As shown in Algorithm 4.2, to evaluate a node for the EVMDD (*k*), the prefetching EVMDD (*k*) machine needs to access the instruction memory only once. On the other hand, as shown in Algorithm 3.1, the standard EVMDD (*k*) machine needs to access the memory twice. The architecture for the prefetching EVMDD (*k*) is simpler than the standard one, since the selector for the indirect branch for the prefetching one is not necessary. Therefore, the prefetching one is faster than the standard one.

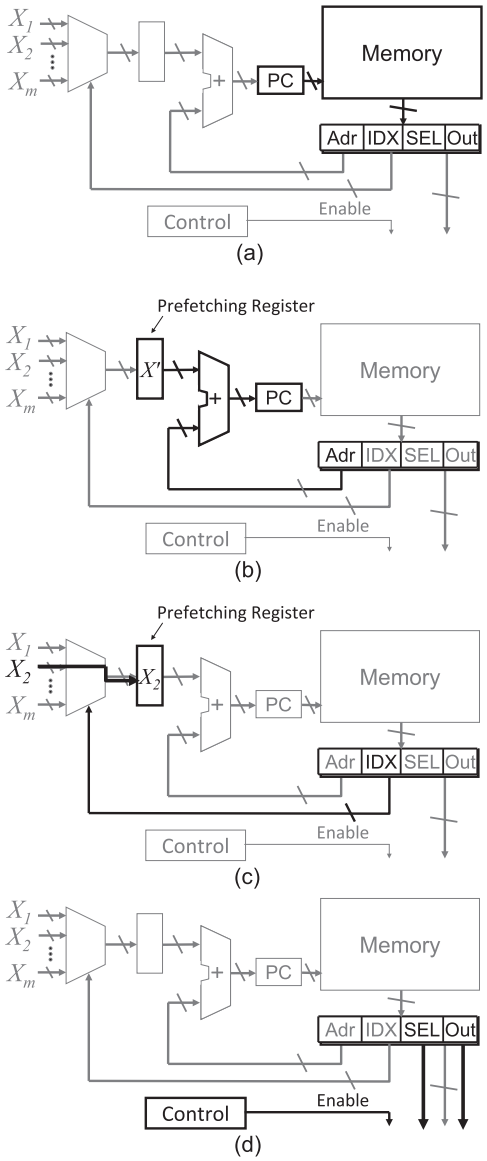

Fig. 14 Operations of the prefetching EVMDD (k) machine.

5. Packet Classifier Using Parallel EVMDD (k) Machine

5.1 Parallel EVMDD (k) Machine

As shown in Sect. 2, a packet classification table is decomposed into five field functions and a rule function. These functions are represented by EVMDDs (k). Figure 15 shows **a parallel EVMDD (k) machine** consisting of six EVMDD (k) machines. The packet header is broadcasted to the inputs of these EVMDD machines by **the Primary Input Bus**. To synchronize six machines, FLAG registers are connected to **the Global FLAG Register (GFLAG)** through bitwise AND gates, and the GFLAG value is broadcasted to the inputs of the EVMDD (k) machines by **the FLAG Bus**. Also, DATA registers are send to

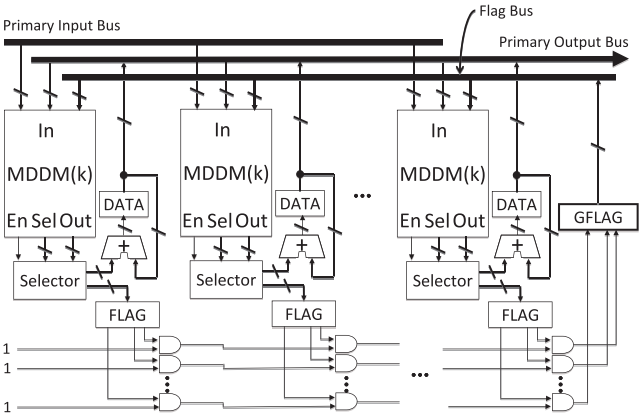

Fig. 15 Parallel EVMDD (k) machine.

other EVMDD (k) machine by **the Primary Output Bus**.

5.2 Partition Rule

The number of nodes in the EVMDD (k) is approximately proportional to the number of segments [18]. The number of segments for a field function is $O(p)$, where p denotes the number of distinct entries. However, that for the rule function is $O(p^5)$. We partition the rule into groups. Since the number of inputs for each group tends to be smaller than that for the whole rules, EVMDD machies for groups is more suitable for parallel computing than that for whole the rules. To partition rules into groups of uniform sizes, this paper proposes the partition method based on **the sizes of segments** as follows:

Definition 5.5: Let $[x, y]$ $(x \leq y)$ be a segment. Then, $y - x + 1$ is the size of the segment.

Since the PRT field has only a constant entry, we partition each field into two groups excluding the PRT field. As shown in Fig. 2, when a field has both a large-size and a small-size segments, the number of segments increases. Thus, we partition the field based on the sizes of segments. To show the size of a segment, we use a binary variable r_i. Let n be the number of bits to represent a segment $[x, y]$. Let $r_i = 0$ for $y - x \leq 2^{n-1}$, and $r_i = 1$ for $y - x > 2^{n-1}$. A group of rules is represented by $\mathcal{G}(r_0, r_1, r_2, r_3)$. As shown in Fig. 16, we assign r_0 to the SA field; r_1 to the DA field; r_2 to the SP field; and r_3 to the DP field.

Algorithm 5.3: (Partition into groups)

1. Partition the rules into groups $\mathcal{G}(r_0, r_1, r_2, r_3)$ based on Fig. 16.
2. Sort $\mathcal{G}(r_0, r_1, r_2, r_3)$ in the descending order of the number of groups.
3. Merge $\mathcal{G}_1(r_0, r_1, r_2, r_3)$ and $\mathcal{G}_2(r'_0, r'_1, r'_2, r'_3)$ those have small number of groups, then generate a group $\mathcal{G}'$.
4. Decompose $\mathcal{G}'$, then represent them by EVMDDs (k).
5. If the number of inputs of an EVMDD (k) for the rule function in $\mathcal{G}'$ is smaller than the total number of inputs of EVMDDs (k) for the rule functions in $\mathcal{G}_1$ and $\mathcal{G}_2$, then go to

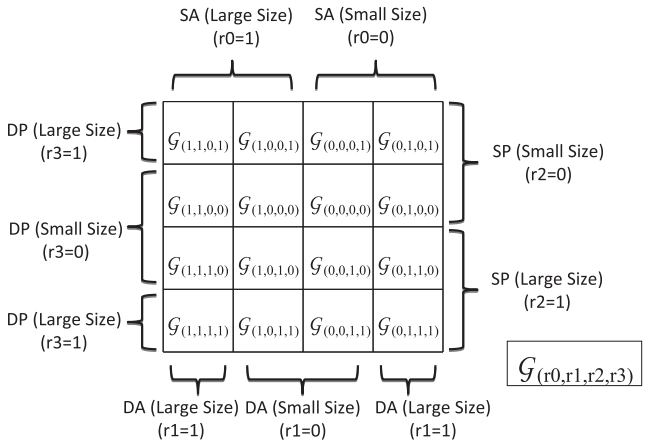

Fig. 16 Partition rule based on the sizes of segments.

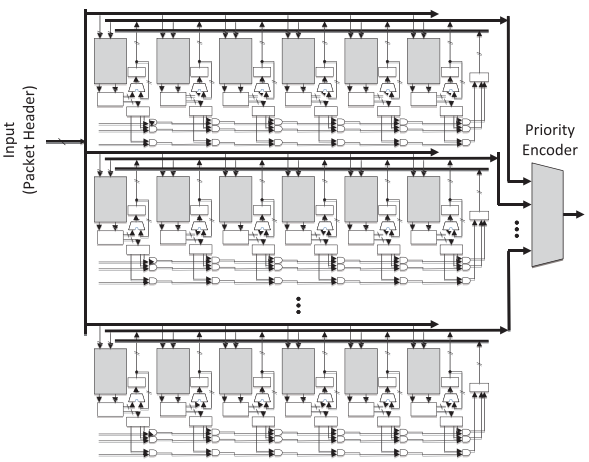

Fig. 17 Overall architecture.

Step 3.
6. Terminate.

5.3 Overall Architecture

Figure 17 shows an overall architecture consisting of parallel EVMDD (k) machines. To synchronize all the EVMDD (k) machines, the GFLAG register is connected to all the EVMDD (k) machines. The packet header is broadcasted to each EVMDD (k) machine. The output of the EVMDD (k) machine is send to the priority encoder to generate the highest rule number.

6. Experimental Results

6.1 Implementation of Parallel EVMDD (k) Machine

We implemented the parallel EVMDD (k) machine on the Avanet Corp. Zedboard (FPGA: Xilinx Inc. Zynq-7020). We used Xilinx Inc. PlanAhead version 14.2 as the synthesis tool. To implement a packet filter, first, we used ClassBench [23] to generate a pseudo packet filter consisting of

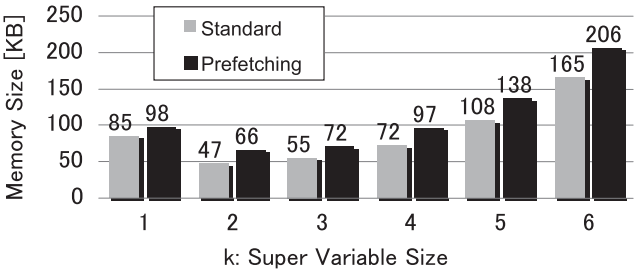

Fig. 18 Comparision of memory sizes [KB].

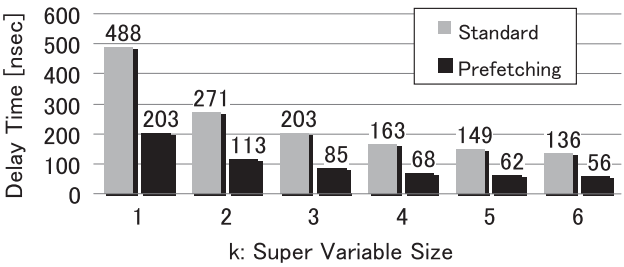

Fig. 19 Comparison of delay times [nsec].

1,000 rules. Then, we loaded the program code for generated EVMDDs (k) into the parallel EVMDD (k) machine. To reduce the number of nodes for the EVMDD (k), we used the sifting algorithm [19]. As for the parallel machine based on the standard EVMDD (k), the maximum clock frequency was 295.10 MHz, while as for that based on the prefetching EVMDD (k), it was 354.14 MHz.

6.2 Comparison with Standard EVMDD (k)

We compared the parallel EVMDD machine (k) based on the standard machine with that based on the prefetching one. Figure 18 compares the memory size [KB] for different sizes k of the super variable. As for $k > 1$, the parallel prefetching EVMDD (k) consumes 1.29 times more memory than the parallel standard EVMDD (k). Figure 19 compares the delay time [nsec] for different sizes k of the super variable. As for the same k, the parallel prefetching EVMDD (k) machine is 2.4 times faster than the parallel standard EVMDD (k) machine. The prefetching EVMDD (k) machine uses only a single instruction, while the standard one uses two instructions. The architecture of the prefetching machine is simpler than the standard one. Thus, the critical path of the prefetching machine is shorter than the standard one. Also, to evaluate a node of the EVMDD (k), the prefetching machine accesses the instruction memory only once, while the standard one accesses twice. Implementation results showed that, as for the clock frequency, the prefetching machine is 1.2 times higher than the standard one. As shown in Figs. 18 and 19, the memory size and the delay time have a trade-off with respect to k. Let A be the area (memory size, in this paper), and T be the delay time. Figure 20 compares the area-delay product (AT^2). Figure 20 shows that the parallel prefetching EVMDD (4) efficiently utilizes the memory size and the

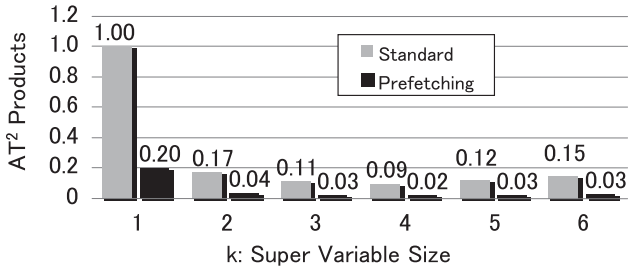

Fig. 20 Comparison of area-delay products (AT^2).

Table 2 Comparison with Intel's Core i5.

Rule	#Grp	Parallel EVMDD (4) Machine (PBM)		Core i5		Ratio (Ci5/PBM)	
		Time [nsec]	Mem [KB]	Time [nsec]	Mem [KB]	Time	Mem
acl1	3	62	115	2938	247	47.4	2.1
acl2	3	62	132	2846	523	45.9	4.0
acl3	3	62	100	2990	355	48.2	3.6
acl4	3	62	119	2940	383	47.4	3.2
acl5	1	62	85	948	333	15.3	3.9
fw1	4	62	34	3806	160	61.4	4.7
fw2	3	62	306	2787	523	45.0	1.7
fw3	4	62	28	3732	248	60.2	8.9
fw4	4	62	27	3742	348	60.4	12.9
fw5	5	62	63	4808	197	77.5	3.1
ipc1	5	62	137	4724	365	76.2	2.7
ipc2	1	62	12	934	149	15.1	12.4

delay time. Although the parallel prefetching EVMDD (4) machine consumes 1.2 times larger memory than the parallel standard EVMDD (4) machine, the prefetching one is 2.4 times faster than the standard one. Therefore, the prefetching machine has a smaller area-delay product than the standard one.

6.3 Comparison with Intel's Core i5 Processor

We compared the delay time and code size for the parallel EVMDD (4) machine with the Intel's general-purpose processor Core i5. We implemented the parallel prefetching EVMDD (4) machine on the Xilinx Zynq-FPGA. It consumed 7,120 look-up tables (LUTs) and 62 block RAMs (BRAMs). Its maximum clock frequency was 354.14 MHz. We used an Intel's Core i5 (2.6GHz at turbo boost mode, Shared cache 3 [MB]), and OS: MacOS X 10.7.5. In the general-purpose processor, the code for the BDD is simpler and can be implemented faster than that for the EVMDD (4) [17]. So, the Core i5 emulates BDDs instead of EVMDDs (4). We generated the execution code by gcc compiler with optimization option -O3. To obtain the execution time for a test vector, we generated random packet headers, and obtained the average time excluding the time for the reading and writing packet headers. Table 2 compares the parallel EVMDD (4) machine with the Core i5 processor with respect to the memory size and the delay time, where *Rule* denotes the name of packet classifier; *#Grp* denotes the number of groups obtained by Algorithm 5.3; *Time* denotes the delay time for a test vector; and *Mem* denotes the memory size. Table 2 shows that, as for the performance, the parallel EVMDD (4) machine is 15.1–77.5 times faster than the Core i5, and as for the memory size, the parallel EVMDD (4) machine requires only 8.1–58.5 percent of the memory of the Core i5.

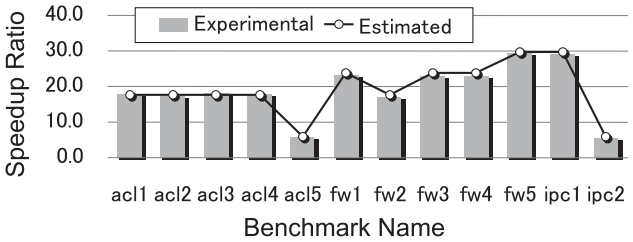

Fig. 21 Speedup ratios for experimental and estimated values.

6.4 Discussion

We analyze the delay time for both the EVMDD (4) machine and the Core i5. The delay time T for the packet classifier using the decision diagram is estimated by

$$T = IPN \times TPI \times PL \times g,$$

where IPN (instructions per a node) denotes the number of instructions to evaluate a node; TPI (time per an instruction) denotes execution time of an instruction; PL (path length) denotes the path length of decision diagram; and g denotes the number of groups obtained by Algorithm 5.3.

As for the parallel EVMDD (4) machine running at 354.14 MHz, $IPN_{EV} = 1$, $TPI_{EV} = \frac{1}{354.14} \times 10^{-6}$, and $PL_{EV} = \frac{n_{rule}}{4}$, where n_{rule} denotes the maximum number of inputs for a rule function. Since the parallel EVMDD (4) machine evaluates all the groups in parallel, $g_{EV} = 1$. As for the Core i5 running at 2.6 GHz, $IPN_{Ci5} = 3$, $TPI_{Ci5} = \frac{1}{2.6} \times 10^{-9}$, $PL_{Ci5} = n_{all}$, and $g_{Ci5} = \#Grp$, where n_{all} denotes the total path length of EVMDDs, and $\#Grp$ is the number of groups shown in Table 2. Although TPI_{Ci5} is shorter than TPI_{EV}, $PL_{Ci5} \times g_{Ci5}$ is longer than $PL_{EV} \times g_{EV}$. In the packet classifier, $PL \times g$ becomes a dominant of the delay time. Figure 21 compares the experimental values with the estimated values with respect to the delay time ratio (Core i5 per the parallel EVMDD (4) machine). From Fig. 21, we can estimate the delay time. As shown in Table 2, the parallel EVMDD (4) machine evaluates the packet classification table much faster than a CPU using its small g and PL.

7. Conclusion

This paper showed the packet classifier based on the parallel prefetching EVMDD (k) machine. First, to represent the packet classification table compactly, rules are partitioned into groups. Then, each group is decomposed into five field functions and a rule function. Next, each function is represented by an EVMDD (k), and it is converted to branch&output instructions. The parallel prefetching EVMDD (k) machine efficiently evaluates the codes for

the packet classification. We selected $k = 4$ for super variable experimentally. We implemented 30 prefetching EVMDD (4) machines on an FPGA, and compared these machines with Intel's Core i5@2.6 GHz microprocessor. As for the performance, the parallel prefetching EVMDD (4) machine is 15.1–77.5 times faster than the Core i5, and as for the memory size, the parallel prefetching EVMDD (4) machine requires 8.1–58.5 percent of the memory of the Core i5. We also analyzed the delay time between the parallel prefetching EVMDD (4) machine and the Intel's Core i5.

The future work is to apply our parallel machine to other decision diagrams (binary moment decision diagrams (BMDs), $*$BMDs, factored edge-valued decision diagrams, etc.).

Acknowledgements

This research is supported in part by the Grants in Aid for Scientistic Research of JSPS, and the Adaptable and Seamless Technology Transfer Program through target-driven R&D of JST. Reviewer's comments were quite useful to improve the presentation.

References

[1] P.C. Baracos, R.D. Hudson, L.J. Vroomen, and P.J.A. Zsombor-Murray, "Advances in binary decision based programmable controllers," IEEE Trans. Ind. Electorn., vol.35, no.3, pp.417–425, Aug. 1988.
[2] R.T. Boute, "The binary-decision machine as programmable controller," Euromicro Newsletter, vol.1, no.2, pp.16–22, 1976.
[3] R.E. Bryant, "Graph-based algorithms for boolean function manipulation," IEEE Trans. Comput., vol.C-35, no.8, pp.677–691, 1986.
[4] CISCO: ASA5500 series, http://www.cisco.com/
[5] E.M. Clarke, K.L. McMillan, X. Zhao, M. Fujita, and J. Yang, "Spectral transforms for large Boolean functions with applications to technology mapping," DAC1993, pp.54–60, 1993.
[6] P. Gupta and N. McKeown, "Packet classification on multiple fields," ACM Sigcomm, August, 1999.
[7] T. Kam, T. Villa, R.K. Brayton, and A.L. Sangiovanni-Vincentelli, "Multi-valued decision diagrams: Theory and applications," Multiple-Valued Logic: An International Journal, vol.4, no.1-2, pp.9–62, 1998.
[8] Y-T. Lai and S. Sastry, "Edge-valued binary decision diagrams for multi-level hierarchical verification," DAC1992, pp.608–613, 1992.
[9] Y-T. Lai, M. Pedram, and S.B. Vrudhula, "EVBDD-based algorithms for linear integer programming, spectral transformation and functional decomposition," IEEE Trans. Comput., vol.13, no.8, pp.959–975, 1994.
[10] C. Meinel and T. Theobald, Algorithms and Data Structures in VLSI Design: OBDD — Foundations and Applications, Springer, 1998.
[11] S. Nagayama and T. Sasao, "Complexities of graph-based representations for elementary functions" IEEE Trans. Comput., vol.58. no.1, pp.106–119, Jan. 2009.
[12] S. Nagayama and T. Sasao, "Representations of elementary functions using edge-valued MDDs," ISMVL2007, 2007.
[13] H. Nakahara, T. Sasao, and M. Matsuura, "A packet classifier using parallel EVMDD (k) machine," MCSoC2013, pp.43–48, 2013.
[14] H. Nakahara, T. Sasao, and M. Matsuura, "A packet classifier using LUT cascades based on EVMDDs(k)," FPL2013, pp.1–6, 2013.
[15] H. Nakahara, T. Sasao, and M. Matsuura, "A comparison of multi-valued and heterogeneous decision diagram machines," J. Multiple-Valued Logic and Soft Computing, vol.19, no.1-3, pp.203–217, 2012.
[16] H. Nakahara, T. Sasao, and M. Matsuura, "On a prefetching heterogeneous MDD machine," MWSCAS2011, pp.1–4, 2011.
[17] H. Nakahara, T. Sasao, M. Matsuura, and Y. Kawamura, "A parallel branching program machine for sequential circuits: Implementation and evaluation," IEICE Trans. Inf. & Syst., vol.E93-D, no.8, pp.2048–2058, Aug. 2010.
[18] H. Nakahara, T. Sasao, and M. Matsuura, "Packet classifier using a parallel branching program machine," DSD2010, pp.745–752, 2010.
[19] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," ICCAD1993, pp.42–47, 1993.
[20] T. Sasao, "On the complexity of classification functions," ISMVL2008, 2008.
[21] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, "Fast and scalable layer four switching," SIGCOM1998, pp.191–202, 1998.
[22] D.E. Taylor, "Survey and taxonomy of packet classification techniques," ACM Computing Curveys, vol.37, no.3, pp.238–275, 2005.
[23] D.E. Taylor and J.S. Turner, "ClassBench: A packet classification benchmark," INFOCOM2005, vol.3, pp.2068–2079, 2005.
[24] P.J.A. Zsombor-Murray, L.J. Vroomen, R.D. Hudson, Le-Ngoc Tho, and P.H. Holck, "Binary-decision-based programmable controllers, Part I-III," IEEE Micro, vol.3, no.4, pp.67–83 (Part I), no.5, pp.16–26 (Part II), no.6, pp.24–39 (Part III), 1983.

Hiroki Nakahara received the B.E., M.E., and Ph.D. degrees in computer science from Kyushu Institute of Technology, Fukuoka, Japan, in 2003, 2005, and 2007, respectively. He has held a research position at Kyushu Institute of Technology, Iizuka, Japan. Now, he is an assistant professor at Kagoshima University, Japan. He received the 8th IEEE/ACM MEMOCODE Design Contest 1st Place Award in 2010, the SASIMI Outstanding Paper Award in 2010, IPSJ Yamashita SIG Research Award in 2011, the 11st FIT Funai Best Paper Award in 2012, the 7th IEEE MCSoC-13 Best Paper Award in 2013, and the ISMVL2013 kenneth C. Smith Early Career Award in 2014, respectively. His research interests include logic synthesis, reconfigurable architecture, digital signal processing and embedded systems. He is a member of the IEEE, the ACM, and the IEICE.

Tsutomu Sasao received the B.E., M.E., and Ph.D. degrees in Electronics Engineering from Osaka University, Osaka Japan, in 1972, 1974, and 1977, respectively. He has held faculty/research positions at Osaka University, Japan, IBM T.J. Watson Research Center, Yorktown Height, NY and the Naval Postgraduate School, Monterey, CA. He has served as the Director of the Center for Microelectronic Systems at the Kyushu Institute of Technology, Iizuka, Japan. Now, he is a Professor of Department of Computer Science, Meiji University, Kawasaki, Japan. His research areas include logic design and switching theory, representations of logic functions, and multiple-valued logic. He has published more than nine books on logic design including, *Logic Synthesis and Optimization*, *Representation of Discrete Functions*, *Switching Theory for Logic Synthesis*, *Logic Synthesis and Verification*, and *Memory-Based Logic Synthesis*, in 1993, 1996, 1999, 2001, and 2011, respectively. He has served Program Chairman for the IEEE International Symposium on Multiple-Valued Logic (ISMVL) many times. Also, he was the Symposium Chairman of the 28th ISMVL held in Fukuoka, Japan in 1998. He received the NIWA Memorial Award in 1979, Takeda Techno-Entrepreneurship Award in 2001, and Distinctive Contribution Awards from IEEE Computer Society MVL-TC for papers presented at ISMVLs in 1986, 1996, 2003 and 2004. He has served an associate editor of the *IEEE Transactions on Computers*. He is a Fellow of the IEEE.

Munehiro Matsuura studied at the Kyushu Institute of Technology from 1983 to 1989. He received the B.E. degree in Natural Sciences from the University of the Air, in Japan, 2003. He has been working as a Technical Assistant at the Kyushu Institute of Technology since 1991. He has implemented several logic design algorithms under the direction of Professor Tsutomu Sasao. His interests include decision diagrams and exclusive-OR based circuit design.